Rational. Synergy

IBM

Classic CLI Help

Rational Synergy Classic CLI Help

Release 7.1

Before using this information, be sure to read the general information under "Notices" on page 528.

This edition applies to VERSION 7.1, Rational Synergy (product number 5724V66) and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

General usage information

This section describes how to use IBM® Rational® Synergy. The following topics are discussed:

- Readme and documentation
- Contacting IBM Rational Software Support
- Using the command line interface - Windows users
- Using the command line interface - UNIX users
- Rational Synergy interfaces
- Rational Synergy help systems
- Terminology and name changes in Rational Synergy 7.1
- Command and argument syntax
- Naming restrictions
- Case and file name limit database options
- Date formats
- Built-In keywords
- Regular expressions
- Administering purposes and templates

Readme and documentation

Be sure to read the latest *Readme* file prior to using or administering Rational Synergy. The Readme file contains much of the information previously contained in the *Release Notes* document, which is no longer issued. The latest *Readme* is located on the Support site (see Contacting IBM Rational Software Support).

The *Rational Synergy README* contains general information about the product release, and includes the following topics:

- System Requirements
- Compatibility with Other Rational Products and Releases
- New Release Features

Other documents referenced in this help can be found on the Documentation section of the DVD or downloaded from Support site.

Contacting IBM Rational Software Support

If the self-help resources have not provided a resolution to your problem, you can contact IBM® Rational® Software Support for assistance in resolving product issues.

> **Note** If you are a heritage Telelogic customer, a single reference site for all support resources is located at http://www.ibm.com/software/rational/support/telelogic/

Prerequisites

To submit your problem to IBM Rational Software Support, you must have an active Passport Advantage® software maintenance agreement. Passport Advantage is the IBM comprehensive software licensing and software maintenance (product upgrades and technical support) offering. You can enroll online in Passport Advantage from http://www.ibm.com/software/lotus/passportadvantage/howtoenroll.html

- To learn more about Passport Advantage, visit the Passport Advantage FAQs at http://www.ibm.com/software/lotus/passportadvantage/brochures_faqs_quickguides.html.
- For further assistance, contact your IBM representative.

To submit your problem online (from the IBM Web site) to IBM Rational Software Support, you must additionally:

- Be a registered user on the IBM Rational Software Support Web site. For details about registering, go to http://www.ibm.com/software/support/.
- Be listed as an authorized caller in the service request tool.

Submitting problems

To submit your problem to IBM Rational Software Support:

1. Determine the business impact of your problem. When you report a problem to IBM, you are asked to supply a severity level. Therefore, you need to understand and assess the business impact of the problem that you are reporting.

   Use the following table to determine the severity level.

Severity	Description
1	The problem has a *critical* business impact: You are unable to use the program, resulting in a critical impact on operations. This condition requires an immediate solution.
2	This problem has a *significant* business impact: The program is usable, but it is severely limited.
3	The problem has *some* business impact: The program is usable, but less significant features (not critical to operations) are unavailable.
4	The problem has *minimal* business impact: The problem causes little impact on operations or a reasonable circumvention to the problem was implemented.

2. Describe your problem and gather background information, When describing a problem to IBM, be as specific as possible. Include all relevant background information so that IBM Rational Software Support specialists can help you solve the problem efficiently. To save time, know the answers to these questions:

   - What software versions were you running when the problem occurred?

   To determine the exact product name and version, use the option applicable to you:

   - Start the IBM Installation Manager and select **File** > **View Installed Packages**. Expand a package group and select a package to see the package name and version number.

   - Start your product, and click **Help** > **About** to see the offering name and version number.

   - What is your operating system and version number (including any service packs or patches)?
   - Do you have logs, traces, and messages that are related to the problem symptoms?
   - Can you recreate the problem? If so, what steps do you perform to recreate the problem?

- Did you make any changes to the system? For example, did you make changes to the hardware, operating system, networking software, or other system components?
- Are you currently using a workaround for the problem? If so, be prepared to describe the workaround when you report the problem.

3. Submit your problem to IBM Rational Software Support. You can submit your problem to IBM Rational Software Support in the following ways:

- **Online**: Go to the IBM Rational Software Support Web site at https://www.ibm.com/software/rational/support/ and in the Rational support task navigator, click **Open Service Request.** Select the electronic problem reporting tool, and open a Problem Management Record (PMR), describing the problem accurately in your own words.

  For more information about opening a service request, go to http://www.ibm.com/software/support/help.html

  You can also open an online service request using the IBM Support Assistant. For more information, go to http://www.ibm.com/software/support/isa/faq.html.
- **By phone**: For the phone number to call in your country or region, go to the IBM directory of worldwide contacts at http://www.ibm.com/planetwide/ and click the name of your country or geographic region.
- **Through your IBM Representative**: If you cannot access IBM Rational Software Support online or by phone, contact your IBM Representative. If necessary, your IBM Representative can open a service request for you. You can find complete contact information for each country at http://www.ibm.com/planetwide/.

Other information

For Rational software product news, events, and other information, visit the IBM Rational Software Web site.

Using the command line interface - Windows users

Rational Synergy supports the command line interface (CLI) under all supported Windows® platforms.

You can execute any Rational Synergy command from the Windows command prompt.

Note that you cannot start the command line interface from the Rational Synergy interface. You must start the CLI separately.

Option delimiter

By default, the Windows client supports the slash (/) option delimiter. The dash (-) option delimiter is also supported. Examples in this help are shown using the dash (-) option delimiter.

Universal naming convention

Use the universal naming convention (UNC) any time you enter a path to an administrative command. UNC makes network access to files, machines, and other devices easier. It enables you to refer to remote machines and files by using a particular format. The format is: `\\`*`computer_name`*`\`*`share_name`*`\`*`path`*.

In the following example, `\\loon\ccmdb\tstgonzo` is a UNC-style path.

```
> ccm message /d \\loon\ccmdb\tstgonzo "Server going down for repair."
```

All Rational Synergy commands accept **both** UNC paths and paths with drive letters (for example, `c:\users\ccmdb\base`). However, three commands, `ccmdb create`, `ccmdb copy`, and `ccmdb unpack` require UNC paths for the database to be created.

File paths

The Windows client supports the standard Windows file specification, which usually is written as:

`drive``:\`*`directory`*`\`*`filename`*

Rational Synergy online help uses the following to represent file paths:

`c:\`*`directory`*`\`*`filename`*

Although your files might reside on a different drive, Rational Synergy help uses drive `c:` for consistency.

Location of CCM_HOME

`CCM_HOME` is the directory where the Rational Synergy product was installed. For example, if you want to edit the `remexec.cfg` file, which resides in the `etc` directory in the Rational Synergy installation area, change directory to *`CCM_HOME`*`\etc`.

The default install directory for a client installation is:
`C:\Program Files\IBM\Rational\Synergy 7.1`

Using the command line interface - UNIX users

Rational Synergy supports the command line interface (CLI) under all UNIX® platforms.

You can execute any Rational Synergy command from the UNIX shell.

Note that you cannot start the command line interface from the Rational Synergy interface. You must start the CLI separately.

Option delimiter

By default, the UNIX client supports the dash (-) option delimiter.

Location of CCM_HOME

`CCM_HOME` is the directory where the Rational Synergy product was installed. For example, if you want to edit the `remexec.cfg` file, which resides in the `etc` directory in the Rational Synergy installation area, change directory to `$CCM_HOME/etc`.

Rational Synergy interfaces

Rational Synergy provides the following interfaces:

- Synergy GUI

  This interface provides full functionality for developers and build managers. It does not support administrative operations, but you can use it for non-administrative use. You can run the Synergy GUI in two modes: Web mode or Traditional mode. These modes are described Web mode and Traditional mode below.

- Synergy CLI

  This interface provides near full functionality for developers and build managers. It does not support administrative operations, and runs in Web mode only. Web Mode is described Web mode and Traditional mode below.

- Synergy Classic GUI

  This interface is available primarily for administrative operations. Users who work with the Classic GUI should plan to switch to the Synergy GUI. The Classic GUI is not available in Web mode. This interface will be phased out in a future release.

- Synergy Classic CLI

  This interface is available primarily for administrative operations, and to provide a transition period for converting existing scripts to use the Synergy CLI. The Classic CLI is not available in Web mode. This interface will be phased out in a future release. You are currently reading the Synergy Classic CLI help.

Web mode and Traditional mode

Synergy 7.1 introduces a new, faster way of working called Web mode. Web mode uses a new underlying architecture for communication between the client and server. It is intended primarily for use across a wide area network (WAN), but can be used on a local area network (LAN) as well. Your Synergy administrator will provide information about which mode you should use.

Web mode and Traditional mode differ in the following ways:

	Traditional mode	**Web mode**
Performance	Same as previous releases	Much faster, especially over a WAN
Synergy GUI	Same as previous releases	Available in Web mode
Classic GUI	Same as 6.5	Not available in Web mode

	Traditional mode	Web mode
Synergy CLI	Not available in Traditional mode	New in Release 7.0. Limited support for administrative commands.
Classic CLI	Same as previous releases	Not available in Web mode
Administration	Same as 6.5	Same as 6.5, plus Synergy server configuration and TDS (see below)
Installation	Same as previous releases	No different from Traditional mode
Network protocol	Proprietary (RFC)	HTTP or HTTPS
User authentication	Operating system (OS)	IBM® Rational® Directory Server™ (TDS), (LDAP)
Work area	Same as previous releases	Supports copy-based work areas only

Eventually Traditional mode will be phased out, but it is supported now because it offers capabilities not yet available through the GUI or CLI in Web mode. You must use Traditional mode for most administration functions, such as saving data offline and cleaning out obsolete data, changing delimiters, adding or modifying type definitions, performing an upgrade, database backups and integrity checks, and migrating data using the migration utility.

Rational Synergy help systems

You are currently viewing help for the Synergy Classic command line interface (CLI) for the 7.1 release of Rational Synergy. This help contains all supported commands, except for some database administration commands that are documented in the *Administration Guides*.

If you use the Synergy Classic graphical user interface (GUI), you may notice similar information in the help system that opens from that interface. Although online help for the CLI is contained in that help system, it is not the most current command line information, as it does not include new commands and options. For the latest commands and options available in this release, start the help while using the command line interface.

The most current help for the Synergy Classic interface is help written for the 6.3 release. The Synergy Classic GUI has had only minor changes since the 6.3 release. Check the README for information about those differences.

If you are not sure which help you are viewing, check the footer on the bottom of each help page, which identifies the release. In addition, help written for Synergy Classic (both the CLI and GUI systems) has a light blue background; help for Rational Synergy has a white background.

Terminology and name changes in Rational Synergy 7.1

The Rational Synergy product is in the process of phasing out certain features and behaviors from the command line and graphical interfaces. This process takes time and has occurred over several releases, in part to give our customers the opportunity to change scripts at convenient times in their work cycles. Depending on which interface you work in, you might see differences in terminology and names.

The following name changes have occurred in the last several releases:

- In releases prior to 6.4, the product was named CM Synergy. The core product was referred to as CM Synergy, with an interface specifically geared for users working in the developer role named CM Synergy for Developers.
- Effective with the 6.4 release, the interface previously named CM Synergy for Developers was enhanced and was referred to as SYNERGY/CM. The other graphical interface was referred to as SYNERGY/CM Classic.
- Effective with the 6.5 release, the interface previously named SYNERGY/CM was enhanced and was referred to as Synergy. The other graphical interface is referred to as Synergy Classic.
- Effective with the 7.0 release, the Rational Synergy interface can be run in two modes: Web mode and Traditional mode. The other graphical interface is still referred to as Synergy Classic. The corresponding command interface is called Synergy Classic CLI.

In addition to the product name changes, some terminology has been changed to be more consistent between interfaces. As this help is intended for use with the Synergy Classic command line interface, the terminology changes listed here affect that interface. The help for the Rational Synergy interface uses the new terminology. Terms used in the synergy classic GUI interface have not changed. The following table shows the terms used in 6.3 and prior releases, the 6.4 release, and current terms used in this CLI help.

Classic GUI and CLI	6.4 Term	Synergy GUI and CLI
Reconfigure/Update Members	Update	Update
Reconfigure Template	Update Template	Process Rule
Reconfigure Properties	Update Properties	Update Properties
Undo Reconfigure	Undo Update	Undo Update
Check Out (Project)	Copy Project	Copy Project
Work Area Snapshot	Copy to File System	Copy to File System
Default Task	Current Task	Current Task

Commands have also been changed for consistency between interfaces. For example, the `ccm update` command also exists as the `ccm reconfigure` command. Links in the help take you to commands renamed with the new terminology. Aliases have been written so that scripts using the 6.3 and 6.4 commands can be used. References to the terms used in prior releases have been placed in the help for your convenience.

Command and argument syntax

You can enter the commands to run the Rational Synergy tool as follows:

- The `ccm` command prefix precedes each user command. Enter user commands individually, such as:

  ```
  Windows:   ccm dir
  UNIX:      ccm ls
  ```

- The Windows administrative commands are discussed in the *Rational Synergy Administration Guide for Windows.* Documents are available on the Rational Software Information Center.
- On UNIX, some administrative commands also use the `ccm` command prefix, but you also must prepend the command with an underscore. Enter these administrative commands individually, such as:

  ```
  $ ccm_install
  ```

- The UNIX administrative commands are discussed in the *Rational Synergy Administration Guide for UNIX*. Users running on an Oracle database should use *Rational Synergy Administration Guide for UNIX (Oracle)*.

The majority of the commands in the command set require a *`file_spec`* or a *`project_spec`* as an argument. These arguments enable you to specify a controlled object in a Rational Synergy database. Other common arguments are *`task_spec`*, *`folder_spec`*, and *`change_request_spec`*.

The following topics are described:

Baseline specification

Change request specification

File specification

Folder specification

Problem specification

Project specification

Project grouping specification

Task specification

Baseline specification

A *baseline_spec* is a reference to baseline name. A *baseline_spec* can have any of the following forms.

- *baseline_name*
- a selection set reference (*@number*)
- the entire selection set reference (@)

When you release a baseline, you can specify a *baseline_spec* that includes a leading DCM database ID (dbid) and a DCM delimiter (for example, J#; where J is the dbid and # is the DCM delimiter).

Version template specification

A *version_template* is any string, with optional keywords, which can be of the form *%keyword* or *%{keyword}*. The keyword can be any Rational Synergy attribute, the special keyword %baseline_name, or the special keywords, %date and %build. When an attribute is specified, the corresponding attribute value from the prep project or product being copied is used.

Alternate keyword syntax for version template

The keyword syntax provides a way to alter the expansion behavior of keywords based on their existence.

- *%{keyword:-string}* If *keyword* is set and is non-null, it expands normally; otherwise it expands to *string*. Note that *string* can be an empty string if you want to see nothing when the keyword is not found.
- *%{keyword:+string}* If *keyword* is set and is non-null, it expands to *string*; otherwise it expands to the empty string (substitute nothing).

So, to get solaris_7.0 or 7.0 (depending on whether platform exists), specify the following:

```
%{platform:-}%{platform:+_}7.0
```

- %{platform:-} expands to solaris if the platform exists (and was solaris); otherwise it expands to the empty string.
- %{platform:+_} expands to _ if the platform exists; otherwise it expands to the empty string.

Change request specification

A *change_request_spec* is a reference to one or more change requests. A *change_request_spec* can have any of the following forms:

- *change_request_number* (a single change request number)
- *change_request_number*,*change_request_number* (a list of change request numbers separated by commas)
- *change_request_number-change_request_number* (a range of values)
- selection set reference (*@number*)

File specification

A *file_spec* is a reference to a non-project object version in the Rational Synergy database.

A *file_spec* can be in one of four forms:

- Work area reference form
- Selection set reference form
- Project reference form
- Object reference form

An *object_name* can contain up to 151 characters; the object's *version* can contain up to 32 characters.

Work area reference form

When an object version is a member of a project and the project is synchronized under a work area directory in the file system, you can reference the object version by its path in the projected directory structure.

The following example uses Windows paths. The functionality is the same on UNIX. For example, if the `foo.c-4` object version is a member of the `jobA-1` project under the `dir1` directory and the project's work area is `c:\users\joe\ccm_tutorial`, the work area reference form for `foo.c-4` is:

```
c:\users\joe\ccm_tutorial\jobA-1\jobA\dir1\foo.c
```

If the current working directory is `c:\users\joe\ccmtest\jobA-1\jobA`, you can reference the `foo.c-4` object version by using the relative path *file_spec*:

```
dir1\foo.c
```

You can augment the *file_spec* with the version to refer to another version of the object:

*path**object_name*[-*version*]

Use this file specification to refer to any different versions of the object version.

For example, if `foo.c-4` is a member of the current project and has a predecessor `foo.c-1`, you can reference the predecessor by using the relative path belonging to `foo.c-4`:

```
dir1\foo.c-1
```

You can use the `ccm delimiter` command to change the delimiter, which separates the object from its version. You must be in the *ccm_admin* role to change the delimiter, as this command changes the delimiter for the entire database. (Be sure to read the Note shown for the delimiter command.)

Selection set reference form

The `ccm query`, `ccm candidates`, and `ccm show` commands display lists of object versions. A list of object versions is called a *selection set*. You can use the object versions in the command line selection set as *`file_specs`*.

For example, enter the following command to list all objects named `foo.c` owned by user *joseph* (the selection set follows the command).

```
ccm query -name foo.c -owner joseph
1) foo.c-1 integrate joseph csrc 1
2) foo.c-2 integrate joseph csrc 1
3) foo.c-3 integrate joseph csrc 1
4) foo.c-4 working joseph csrc 1
```

To use the command line selection set reference form to receive information on `foo.c-2`, use the `ccm properties` command as follows:

```
ccm properties @2
```

For some commands, you can reference the entire contents of the command line selection set by using the @ symbol only:

```
ccm properties @
```

Project reference form

When an object version is a member of a project, but the project is not under a work area directory in the file system, you can reference the object version by its relative path within the project. This example uses Windows paths:

`relative_path``\`*`object_name`*`[-`*`version`*`]@`*`project_name-project_version`*

For example, to use the project reference form for the `foo.c-4` object version in the `jobA-1` project in the `dir1` directory, use the following *`file_spec`*:

```
jobA\dir1\foo.c@jobA-1
```

You do not need to know the version of `foo.c` that you are referencing; you only need to know the name of the project in which it is a member.

> **Note** In a DCM database, the *`project_spec`* must be a four-part name (*`object_name-version`*`:`*`type`*`:`*`instance`*). To learn more about the *`project_spec`*, see Project specification.

Object reference form

The object reference form (also called the four-part name) requires the name, version, type, and instance of the object version. The object reference form is:

object_name-version:*type*:*instance* or
object_name:version:*type*:*instance*

For example, use the following object reference form for the second instance of the `foo.c-4` object version, which is a `csrc` object:

```
foo.c-4:csrc:2
```

> **Note** You and other users could have several objects with the same name and type (for example, multiple `csrc` files named `main.c`). The instance differentiates objects with the same name and type.

Folder specification

A *folder_spec* is a reference to one folder number. A *folder_spec* can have any of the following forms.

- *folder_number* (an integer value)
- DCM-style *folder_number* (*database_id*#*folder_number*)
- four-part name (*object_name-version*:*type*:*instance*)
- selection set reference (*@number*)
- name of a file that contains a **single** folder number (*folder_id_filename*)

When the *folder_specs* reference is to more than one folder number, the *folder_specs* can have any of the following forms:

- *folder_specs*,*folder_specs* (a list of values)
- *folder_specs*-*folder_specs* (a range of values)
- name of a file that contains **one or more** folder numbers (*folder_id_filename*)

The *folder_spec* used in the *folder_specs* (multiple argument) syntax must be in the *folder_number*, DCM *folder_number*, or selection set form.

For example, to add the `irv` database's 14th folder and the current database's (not `irv`'s) sixth, seventh, and eighth folders to the `jobA-1` project's update properties, use the following *folder_specs*:

```
ccm up -a -folders irv#14,6-8 jobA-1
```

Folder template specification

A *folder_template_spec* is a reference to one folder template. A *folder_template_spec* can have any of the following forms:

- *folder_template* (a name such as "All completed tasks for release %release")
- DCM-style *folder_template* (*database_id*#*folder_template*)
- four-part name (*object_name-version*:*type*:*instance*)
- selection set reference (*@number*)
- name of a file that contains a **single** folder name (*folder_template_filename*)

When the *folder_template_specs* reference is to more than one folder template, the *folder_template_specs* can have any of the following forms:

- *folder_template_specs*,*folder_template_specs* (a list of values)
- name of a file that contains **one or more** folder descriptions (*folder_template_description_filename*)

The *folder_template_spec* used in the *folder_template_specs* (multiple argument) syntax must be in the *folder_template*, DCM *folder_template*, or selection set form.

For example, to add the `irv` database's ninth folder template and the current database's (not `irv`'s) fourth folder template to the process rules for the "Integration Testing" purpose for release 3.2, use the following *`folder_template_specs`*:

```
ccm process_rules -add -folder_temps irv#T9,T4 3.2:"Integration
Testing"
```

Process rule specification

A *`process_rule_spec`* is a reference to a process rule. A *`process_rule_spec`* can have any of the following forms:

- selection set reference (*`@number`*)
- four-part name
- *`release`*`:`*`name of generic process rule`* for release-specific rules
  `defined_name` for generic process rules

> **Note** The old form of *`update_template_spec`* is no longer accepted. If used, it will fail, unless the name of the purpose specified is the same name as a generic process rule.

For example, to add a folder template, `T7`, to the release-specific process rule `1.0: Collaborative Development`, use the following *`process_rule_spec`*:

```
ccm process_rule -add -folder_temp T7 1.0:"Colloborative Development"
```

Problem specification

A *problem_spec* is a reference to one or more problems. As problems are now called change requests, usage of *problem_spec* will be replaced by *change_request_spec* in a future release. A *problem_spec* can have any of the following forms:

- *problem_number* (a single problem number)
- *problem_number*,*problem_number* (a list of values)
- *problem_number*-*problem_number* (a range of values)
- selection set reference (*@number*)

Project specification

A *project_spec* is a reference to a project object version.

When a command requires a project as an argument, you must supply a *project_spec* to the command. In a DCM database, the *project_spec* must be a four-part name (*object_name-version*:*type*:*instance*) if the project doesn't have the default instance of dbid#1, where dbid is the database's database identifier. For projects that were created in the current database with multiple non-local project instances disabled (the default setting), projects can be specified with a two-part spec. Otherwise, the *project_spec* consists of the name and the version of the project:

project_name-project_version

For example, to show information about the jobA-1 project, use the following *project_spec*:

```
ccm properties -p jobA-1
```

A *project_name* can contain up to 155 characters; the project's *version* can contain up to 32 characters.

The default delimiter is - (hyphen). To learn how to change your delimiter, see delimiter command.

You can also use the following form to parse a *project_spec*.

project_name version_delimiter : project_version : *project_instance*

If you specify the two-part form (that is, without the project instance), a default instance is assumed, as follows:

- In a non DCM-initialized database: 1
- In a DCM-initialized database: *dbid dcm_delimiter* 1 (for example, A#1)

Project grouping specification

A *project_grouping_spec* is a reference to a project grouping. Grouping of projects allows you to update a group of projects using the same baseline and tasks.

A *project_grouping_spec* can be any of the following:

- The project grouping's four-part name. The name consists of:

  - The *name* attribute is the same ASCII encoding of the `release` and `member_status` that is used for the name of a process rule.

  - The *version* attribute, which is always `1`.

  - The *type*, which is always `project_grouping`

  - The *instance* is either:
  A private project grouping's instance is the same as its owner, for a non-DCM-enabled database. For a DCM-enabled database, the instance is the local database id, followed by the DCM delimiter, followed by the owner

  or:

  - A non-private project grouping's instance is `1`, for a non-DCM-enabled database. For a DCM-enabled database, the instance is the database id of the database in which it was created, followed by the DCM delimiter, followed by `1`

  An example of the four-part name is:
  `CM%002f7.1%3Acollaborative-1:project_grouping:linda`

- The default displayname for a project grouping. For cases where the user is the owner of the project grouping, *user_name's* is allowed as an alternative to `My`. If the displayname for project_grouping is customized, it does not affect the syntax of the *project_grouping_spec*.

  Examples of the displayname are: `Linda's Synergy/7.1 Collaborative Development Projects` and `All Synergy/7.1 Integration Testing Projects for Database P`

- A query reference (*@number*) that refers to a project grouping in a selection set from a query, such as `@1`
- For commands that allow multiple project groupings, a query reference that refers to the entire selection set, such as: `@`

Task specification

A *task_spec* is a reference to one task number. A *task_spec* can have any of the following forms:

- *task_number* (an integer value)
- DCM-style *task_number* (*database_id*#*task_number*)
- four-part name (*object_name-version*:*type*:*instance*)
- selection set reference (*@number*)
- name of a file that contains a *single* task number (*task_id_filename*)

When the *task_specs* reference is to more than one task number, the *task_specs* can have any of the following forms:

- *task_specs*,*task_specs* (a list of values)
- *task_specs*-*task_specs* (a range of values)
- name of a file that contains a **one or more** task numbers (*task_id_filename*)

> **Note** task_specs that reference multiple specs can be separated by a comma and white space.

The *task_spec* used in the *task_specs* (multiple argument) syntax must be in the *task_number*, DCM *task_number*, or selection set form.

For example, to add the current database's fifth, eighth, ninth, and tenth tasks in the query output to the jobA-1 project's update properties, use the following *task_specs*:

```
ccm up -a -tasks @5,@8-@10 jobA-1
```

Naming restrictions

This section describes the Rational Synergy object, release, database, and DCM naming restrictions.

Restricted object names

An object name can contain any combination of alpha numerics and symbols **except** for those characters that are restricted.

You cannot use any of the restricted characters as a version delimiter. For more information, see delimiter command.

Following are some of the Rational Synergy object naming restrictions.

- 8-bit and double-byte characters (with the top bit set) are not permitted in object names.
- Project names must not contain tabs. Makefile names must not contain tabs or spaces.
- Keyword expansion in source files whose names include spaces may contain syntax errors when compiled. Avoid using spaces in source code file names, or comment out (or remove) the keyword.

Other restricted characters, and the reasons they are restricted, are shown in the following table.

Character	Why restricted
/	UNIX path delimiter; internal delimiter
\	Windows path delimiter; escape character
'	UNIX quoting character (forward quote)
"	Windows quoting character
:	Windows drive letter delimiter; Rational Synergy object specification delimiter
?	INFORMIX single-character wild card; regular expression
*	INFORMIX multiple-character wild card; regular expression
[	INFORMIX match syntax; regular expression
]	INFORMIX match syntax; regular expression
@	Rational Synergy object specifications delimiter
-	Rational Synergy version delimiter

You cannot use the following characters as the first character in an object name:

- , (comma)
- + (plus sign)
- - (dash)
- ~ (tilde)

Restricted release names

Each Rational Synergy release name must conform to the following conventions:

- The release name cannot contain any of the restricted characters shown in the preceding table.
- The values `Any`, `None`, `none`, `as_is`, and `Default Release` are reserved values that cannot be used.
- The component name must contain 64 or fewer characters.
- The component release must contain 32 or fewer characters.

Restricted database names

Each Rational Synergy database name must conform to the following conventions:

- If two databases use the same database server, they cannot have the same name. The name is the leaf directory in the full database path.
- The database name can contain letters, digits, and underscores only.
- The database name must begin with a letter.
- The database name must contain 18 characters or fewer.

> **Note** When naming a Rational Synergy database, uppercase and lowercase characters are equivalent.

Restricted baseline names

Each Rational Synergy baseline name must conform to the following conventions:

- The name cannot contain the # character.

DCM restrictions

Following are naming restrictions for DCM databases.

The following characters cannot be used in the database ID of the DCM database.

Character	Why restricted
/	UNIX path delimiter; internal delimiter
\	Windows path delimiter; escape character
'	UNIX quoting character (forward quote)
"	Windows quoting character
:	Windows drive letter delimiter; Rational Synergy object specification delimiter
$	INFORMIX single-character wild card; regular expression
?	INFORMIX single-character wild card; regular expression
*	INFORMIX multiple-character wild card; regular expression
[	INFORMIX match syntax; regular expression
]	INFORMIX match syntax; regular expression
@	Rational Synergy object specifications delimiter
<space>	The database ID cannot be enclosed in quotes
#	Rational Synergy DCM delimiter; comment in GNU makefiles

In addition to the restricted characters listed above, the database ID cannot be longer than 8 characters and cannot be the name "probtrac". The database ID cannot contain the version delimiter by default. This can be changed by the allow_delimiter_in_name attribute.

The DCM database ID is case-sensitive. In any DCM cluster that uses lowercase databases, DCM database IDs must be unique without respect to case. That is, you must not use database IDs that are only different with regards to case.

You cannot use characters a-z, A-Z, or 0-9 for the DCM delimiter. You can use "!", "~", or "=" as an alternative delimiter. The default DCM delimiter is "#", which should be used whenever possible.

Case and file name limit database options

The following two database options, Case and File name limit, could have an impact on the names you give your objects in the Rational Synergy database:

Case

Rational Synergy supports case-sensitive file names. The keywords that support this option enable you to preserve the case of object names or to make object names lowercase in a Rational Synergy database.

If you want to view the case setting for your database, enter the following command:

```
ccmdb info database_path [-k case]
```

For a discussion of how to change the case option, refer to the `ccmdb_info` command in the appropriate Rational Synergy *Administration Guide*.

File name limit

File name limits are dependent on both file system and Rational Synergy limitations. By default, you can create objects (files, directories, and projects) with names up to 256 characters (Windows) or 155 characters (UNIX) in a Rational Synergy database. (See Command and argument syntax for a list of illegal symbols.)

To view the file name limit keyword for your database, enter the following command:

```
ccmdb info database_path [-k filelimit]
```

To change the file name limit keyword, you must be working as user *ccm_root*. For a complete discussion of how to change the file name limit mode, refer to the `ccmdb info` command in the appropriate *Rational Synergy Administration Guide*.

Date formats

For more information on acceptable date formats in Rational Synergy, see Date formats.

Built-In keywords

The following keywords are built into Rational Synergy. You can use these keywords to control the format of the output from query, list, and show operations on the command line, and query operations in the GUI.

> **Note** You also can use attribute names as keywords. To list the attributes that are associated with an object, use the `ccm attr` command with the `-list` option.

Keyword	Description
`%baseline`	Returns a project's baseline project. Returns `<No baseline>` if no baseline exists.
`%change_request`	Displays one or more change requests that are associated with the object. For a file, these change requests are determined based on the associated tasks and the change requests that are associated with those tasks.
`%change_request_duplicates`	Returns a list of a change request's duplicate change requests.
`%change_request_original`	For a change request in the duplicate state, returns the original change request of which it is a duplicate.
`%change_request_release`	Displays the release property of change requests that are associated with the object.
`%change_request_status`	Displays the status of one or more change requests that are associated with the object.
`%change_request_synopsis`	Displays the synopsis of one or more change requests that are associated with the object.
`%dcm_delimiter`	Returns a number sign (#) for non DCM-initialized databases. Returns the actual DCM delimiter for DCM-initialized databases.
`%displayname`	Defaults to *`name-version`*.
`%fullname`	Returns the four-part name in *`subsystem/cvtype/name/version`* format.
`%has_`*`relationship`*	Displays those objects that have a relationship from the object in the query.

Keyword	Description (Continued)
%in_baseline	Returns the name of a project's baseline, if the project is in a baseline.
%in_build	Returns the build number of the baseline for projects that are members of a baseline, if the project is in a baseline.
%instance	Alias for the %subsystem part of the object name.
%is_*relationship*_of	Displays those objects that have a relationship to the object in the query.
%model	Returns the %fullname of the current model object.
%objectname	Returns object name in *name-version*:*cvtype*:*subsystem* format.
%optional_project_instance	Returns a blank string if it has a default instance value ("1"). Returns a DCM delimiter and project instance for non-default ("*dbid*#1").
%problem_duplicates	Returns a list of a problem's duplicate problems.
%problem_original	For a problem in the duplicate state, returns the original problem of which it is a duplicate.
%purpose	Displays a project's purpose.
%requirement_id	Displays the requirement ID saved on the change requests that are associated with the task or object's associated tasks.
%root	Returns the <no_root> string if the object is not a root directory, and the project's %fullname if it is a root directory.
%sourcename	Defaults to the name of the object.
%states	Returns legal object states separated by spaces.

Keyword	Description (Continued)
%task	Returns a comma-separated list of task numbers associated with this object. Returns <void> if no associated tasks exist.
%task_platform	Returns a comma-separated list of platform values of the tasks associated with this object. Returns <void> if no associated tasks exist.
%task_release	Returns a comma-separated list of release values of the tasks associated with this object. Returns <void> if no associated tasks exist.
%task_status	Returns a comma-separated list of task statuses associated with this object. Returns <void> if no associated tasks exist.
%task_subsystem	Returns a comma-separated list of subsystems (task_subsys) values of the tasks associated with this object. Returns <void> if no associated tasks exist.
%task_synopsis	If the object is a task, returns the task_synopsis attribute. Otherwise, returns a semi-colon-separated list of task_synopses for the tasks associated with this object, or <void> if no tasks are associated.
%type	Returns the type of the object (stored in the cvtype attribute).

Regular expressions

The following regular expressions can be used in certain commands to match against string values and optionally specify replacements to be made in the resulting string

- Ordinary characters

  An ordinary character in a regular expression matches itself. The ordinary characters are characters other than those described below as special characters.

  ```
  (  )  [  ]  ^  $  .  *  +  ?  |  \
  ```

- Special characters

  The special characters affect the matching behavior of regular expressions as described in the table below. Note that constructs that match arbitrary-length character sequences, i.e., `* + ?`, will always match the longest left-most string that permits a match.

  The table below shows special characters and their restrictions.

Character	Why restricted
`^`	Matches the beginning of the string. For example: `str ? * "^abc"` only matches if `str` starts with `abc`
`$`	Matches the end of the string. For example: `str ? * "abc$"` only matches if `str ends` with `abc`
`.`	Matches any single character. For example: `str ? * "a.c$"` matches values of `str` containing `abc, axc,` etc
`*`	Matches zero or more of the immediately preceding expression. For example: `str ? * "ab*c$"` matches values of `str` containing `ac, abc, abbc,` etc
`+`	Matches one or more of the immediately preceding expression. For example: `str ? * "a+c$"` matches values of `str` containing `abc, abbc, agggc,` but not `ac`
`?`	Matches zero or one of the immediately preceding expression. For example: `str ? * "ab?c"` only matches if `str contains ac` or `abc`

Character	Why restricted
|	Matches either the preceding or following expression. For example: `str ? * "a|b|c"` matches values of `str` containing `a, b,` or `c`
`[ ]`	Matches any single character listed between brackets. For example: `str ? * "[ab]c"` matches values of `str` containing `ac or bc`
`[^ ]`	This combination of characters matches any single character not listed between brackets. For example: `str ? * "a[^b]c"` matches values of `str` containing `axc` for any replacement of `x` except for `b`
\	Escapes the character which immediately follows. For example: `str ? * "a\.c$"` matches if `str` contains `a.c, and str ? * "a\\c$"` matches if `str` contains `a\c` To embed a backslash character in a string, the string literal must contain two consecutive backslashes.
`( )`	Delimits subexpressions. For example: `str ? * "a(b|c)*d*"` matches if `str` contains `a` followed by any number of `b`'s or `c`'s followed by `d`, such as `"ad"` or `"acbbccd"`

Wild card match regular expressions

The following characters can be used with the keyword MATCHES.

Character	Why Restricted
*	Matches zero or more characters
?	Matches any single character
\	Removes the special significance of the next character (used to match* or ? by writing * or \?)

Administering purposes and templates

The CM administrator (the *ccm_admin* role) can define the roles allowed to perform administrative operations for project purposes and process rules.

Project purpose manager

A project purpose manager is any user with a role that contains the privilege PRIVILEGE_MANAGE_PROJECT_PURPOSES. By default, the privilege is contained in two roles: *build_mgr* and *ccm_admin.* Each site can add or remove this privilege from any role.

A project purpose manager can create or delete the project purposes for a database. However, if a build manager tries to modify a purpose that would require modification of a project for which the build manager does not have permission to modify, the operation will fail.

Only a user in the *ccm_admin* role can edit this privilege.

Process rules manager

The process rules manager (previously called reconfigure properties and update template manager) exists for databases that use process rules.

A process rules manager is any user with a role that contains the privilege PRIVILEGE_MANAGE_PROCESS_RULES. By default, this privilege is contained by two roles: *build_mgr* and *ccm_admin,* and only a user in the *ccm_admin* role can edit this privilege. Each site can add or remove this privilege from any role.

A process rules manager can create or edit a process rule. However, if a build manager tries to modify a process rule that would require modification of a project for which the build manager does not have permission to modify, the operation will fail. In addition, a build manager cannot delete a process rule that is in use in any developer's working project. Only a user in the *ccm_admin* role can delete a process rule.

For information about setting roles, see role_definitions.

Release manager

The release manager is any user with a role that contains the privilege PRIVILEGE_MANAGE_RELEASES. By default, this privilege is contained by two roles: *build_mgr* and *ccm_admin*. Each site can add or remove this privilege from any role.

A release manager can create or edit release information. However, some operations require the *ccm_admin* role, for example, renaming or deleting a release that is in use.

Default settings

In order for Rational Synergy to be operational when you install it, it is shipped with a set of pre-designed values or settings, called defaults. These default settings have been defined as the settings that a majority of users would choose. However, these settings can be modified by you to better meet your needs. The information presented here defines the default values and where they are stored, describes how to change them, and explains any interaction between the settings. The following topics are discussed:

- How defaults are set
- Default options
- Initialization file - Windows
- Initialization file - UNIX
- Startup file
- GUI settings
- Environment variables

How defaults are set

The typical ways you can set or change default values are as follows:

- System-wide settings
- Database-wide settings
- Personal settings
- Command line settings

Rational Synergy reads the system-wide or database-wide settings first, then the personal settings, and then any values set from the command line. The last values that are read override the previous settings. The following paragraphs describe these common ways of setting default values.

System-wide settings

System-wide default settings affect all users of an installation area. These defaults are usually set in the *system initialization* (`ini`) file.

The initialization file is called `ccm.ini`, and is found in the `etc` directory in *`CCM_HOME`*. All users of the installation area must restart their sessions to be able to use a new default setting in the system initialization file.

Database-wide settings

Database-wide settings affect all users of a specific database. These defaults are usually set in an attribute on the model object, or in an attribute on a specific type object. When you change a setting by modifying an attribute in the model, you may need to restart your session for the new setting to take effect.

Personal settings

Personal settings affect only your own sessions and databases. You set these defaults in one of three places, depending on the particular option:

- In the `[Options]` section in your *personal initialization* (`ini`) file

  On Windows, the initialization file is called `ccm.ini`, and you create it in your Windows Documents and Settings directory (for example, `C:\Documents and Settings\`*`user_name_directory`*).

  On UNIX, the initialization file is called `.ccm.ini`, and you create it in your `$HOME` directory.

  You must restart your session to be able to use a new default setting in your initialization file.

- On the command line

  Some personal settings are set using the command line. Some of these options are also available from the Rational Synergy **Options** dialog box.

- In the Rational Synergy **Options** dialog box

  Some options are set in the **Options** dialog box. Some of these options may also be set on the command line.

Command line settings

You can use the `ccm set` command to set many Rational Synergy options by setting variables from the command line. This action sets the defaults for your immediate use so that you do not need to restart your session to have take effect. Depending on the option, the setting may apply for your current session only, or may persist between sessions. The syntax of the `set` command is:

`ccm set` *`variable_name variable_value`*

Most of the options that can be set with the `ccm set` command apply for the current session only. Those that are persistent are marked as such.

Default options

The following section contains the Rational Synergy options, their default values, and where to set them. The options are listed in alphabetical order. The option names are case-*insensitive,* when specified as a variable on the command line or in a personal initialization file. However, for those options that are specified in a model attribute, the name of the attribute must be in lower case.

Rational Synergy uses settings made to the default options in the following order of precedence:

1. On the system or database level (that is, the system `ccm.ini` file or model attribute)

   Rational Synergy reads the Options set in the system `ini` file or the appropriate model attribute first.

2. On the personal level (that is, in your personal `ccm.ini` file)

   Options set in the personal `ini` file override system `ini`-level settings.

3. Using the `ccm set` command

   Changes made using the `ccm set` command override both the system and personal `ini`-level settings.

The default line continuation character in an initialization file is a plus sign (+) on Windows and a backslash sign (\ ) on UNIX.

For information about how to set model attribute options, see Setting model object attribute options.

For information about how to set options in the system or personal `ini` file, see Setting options in the system or personal ini file.

For information about how to set options using the set command, see Setting options using the ccm set command.

add_object_task_assoc

Set Option: Model object attribute

Ensures that an existing object being added to a project is associated with the current (default) task. This option is used with the Paste operation (from the GUI) or the use command (from the CLI).

The default is `TRUE`.

If you want your model to match Rational Synergy release 4.5 or earlier, you must set this option to `FALSE`.

You must restart your session for this change to take affect. For information about how to set model attribute options, see Setting model object attribute options.

allow_delimiter_in_name

Set Option: Model object or type-specific attribute

Controls whether the delimiter is a restricted character.

When set to `TRUE`, the current delimiter is no longer a restricted character for non-project object names. The delimiter is still restricted for versions, types, instances, and projects.

With this feature enabled, object parsing is done from right to left in the sense that the right-most delimiter character is taken to be the delimiter. As a practical matter, object identification is done by first attempting to identify a given string as a name, and if that fails, it is identified as *name<delimiter>version*. This capability is particularly an issue with create, move, and use.

> **Note** All databases in the same DCM cluster **must** use the same value for this attribute. Failure to synchronize this value may result in undesirable behavior similar to having objects with "~" in them, and changing the delimiter to a "~."

With the feature enabled, you can also create non-project objects with versions. However, you will also be unable to use `ccm move` to set a version on a renamed file. (You can work around this limitation by using the `ccm attr` command or the Properties dialog to change the version.)

This attribute can also exist on individual types. In this case, the database setting is overridden if the database setting is `FALSE` and the type-specific setting is `TRUE`.

The following built-in types have `allow_delimiter_in_name` set to `TRUE`:

```
process_rule
processdef
saved_query
releasedef
project_grouping
folder_temp
```

The default is `FALSE`.

This option has the following restrictions and effects:

- Project names cannot contain the delimiter. If a user attempts to create or migrate a project whose name contains the delimiter, it will fail with an error message.
- After turning on this option, the version can no longer be specified for the create operation in the GUI or CLI; it will always treat the `object_spec` as the name. (Before this change, you can specify both the name and the version when creating an object, for example, specifying `foo-one` would create an object named `foo` with version `one`. After the delimiter change, it will create an object named `foo-one` with version `1`.) Otherwise, there is no other way to create an object whose name contains the delimiter; you would need to create it and then rename it.
- After turning on this option, CLI commands that use the object reference form *name<delim>version* will first try to find an object with that name, and if that fails, will

try again without the part to the right of the delimiter. For example, if files named `foo-one` and `foo` both exist in the work area, and you specify `foo-one`, it will first look for a file named `foo-one`. Only if a file with that name is not found will it look for a file named `foo` with version `one`. You can still identify the other file (`foo` version `one`) using its 4-part name or the selection set reference form.

- After turning on this option, CLI commands will fail for objects that have the delimiter as the first character of the name, if the delimiter is `-` (dash or minus) because the delimiter is also the option delimiter. For example, the command `ccm create -foo.c` will fail.

For information about how to set model attribute options, see Setting model object attribute options.

allow_prep

Set Option: Project or project type attribute.

Enables you to include *prep* subprojects when you update your project, if they are candidates (either from using a prep project as a baseline or from including in your update properties a task or folder that contains them).

This option is provided to support alternative methodologies. However, if you use this option, you run the risk of overwriting prep products with different (inappropriate) contents. Here is what can happen:

-- If a project includes *prep* subprojects, and the project owner (or build manager, in a *prep* project) is running in the *build_mgr* role, then when he builds his project, the projects within the *prep* subprojects can be rebuilt if they are determined to be out of date. After being rebuilt, they may be out of sync with respect to the rest of the products that make up the software for which they were last built.

The default is `FALSE`, and *prep* subprojects cannot be used when a project is updated.

A project type attribute is set the same way as model attributes. For information about how to set model attribute options, see Setting model object attribute options.

baseline_template

Set Option: Model object attribute or `ccm set` command or **Options** dialog box

Specifies the version template to be used for project and products in a baseline when none is explicitly specified in the create baseline or modify baseline operation.

The default is `%{version}_%date`

Use the `ccm set` command to change the template to be used. The setting is persistent and applies to all sessions on all clients for the given user in the given database.

This option is available in the **Options** dialog box. This setting is also persistent and applies to all sessions on all clients for the given user in the given database.

The syntax for a baseline template is defined in the baseline command.

For information about how to set model attribute options, see Setting model object attribute options.

For information about how to set options using the set command, see Setting options using the ccm set command.

baseline_template_date_format

Set Option: Model object attribute or `ccm set` command or **Options** dialog box

Specifies the date format to be used when creating a baseline when expanding the `date` keyword in the `baseline_template`.

The default is `= %Y%m%d`

Use the `ccm set` command to change the date format to be used. The setting applies to all sessions on all clients for the given user in the given database.

This option is available in the **Options** dialog box. This setting is also persistent and applies to all sessions on all clients for the given user in the given database.

For information about how to set model attribute options, see Setting model object attribute options.

For information about how to set options using the set command, see Setting options using the ccm set command.

baseline_template_repl_char

Set Option: Model object attribute or `ccm set` command or **Options** dialog box

Sets the default version string replacement character that is used if the instantiated *`version_template`* for any project or product in the baseline contains characters that are not allowed in a version string.

The default is the underscore character (_).

For example, if `%platform` is part of a project version template, and the prep project has a platform of `SPARC-solaris`, then the version string contains the string `SPARC_solaris`. Or, if `%release` is part of a product version template, and the prep product has a release of `CM/7.1`, then the version string contains the string `CM_7.1`.

Use the `ccm set` command to change the character to be used. The setting applies to all sessions on all clients for the given user in the given database.

This option is available in the **Options** dialog box. This setting is also persistent and applies to all sessions on all clients for the given user in the given database.

For information about how to set model attribute options, see Setting model object attribute options.

For information about how to set options using the set command, see Setting options using the ccm set command.

check_release

Set Option: Model object attribute

Compares the release values of an object and its associated task to ensure they are the same. If the values do not match, a message is written to the Message View (`ccm_ui.log`) informing you of the mismatch.

The default is `TRUE`.

For information about how to set model attribute options, see Setting model object attribute options.

cli_compare_cmd
cli_proj_compare_cmd
cli_dir_compare_cmd
cli_symlink_compare_cmd
cli_merge_cli
cli_dir_merge_cmd

Set Option: System or personal `ini` file, or object, or object type attribute or `ccm set` command

`cli_compare_cmd` is the default command that is executed when two normal files are compared from the CLI. A normal file is an object that is not a project, directory, or symbolic link. It defaults to the `cli_compare_cmd` attribute on the first object selected, which is, by default, `ccm_dff -o %outfile %file1 %file2`.

`cli_proj_compare_cmd` is the default command that is executed when two projects are compared from the CLI. It defaults to the `cli_compare_cmd` attribute on the first project selected, which is, by default, `sdiff -w 80 %file1 %file2`.

`cli_dir_compare_cmd` is the default command that is executed when two directories are compared from the CLI. It defaults to the `cli_compare_cmd` attribute on the first directory, which is, by default, `%ccm_merge`.

`cli_symlink_compare_cmd` is the default command that is executed when two symlinks are compared from the CLI. It defaults to the `cli_compare_cmd` attribute on the first symlink, which is, by default, `ccm_dff -o %outfile %file1 %file2`.

`cli_merge_cmd` is the default command that is executed when two normal files are merged from the CLI. A normal file is an object that is not a project, directory, or symbolic link. It defaults to the `cli_compare_cmd` attribute on the first object selected, which is, by default, `%ccm_merge`.

`cli_dir_merge_cmd` is the default command that is executed when two directories are merged from the CLI. It defaults to the `cli_compare_cmd` attribute on the first directory, which is, by default, `%ccm_merge_dir`.

On Windows, defaults for all four of these options are specified on the system initialization file, overriding the defaults on the object types. The Window defaults for all four options are the same: `ccm_dff -o %outfile %file1 %file2`

All lines in your initialization file that reference the `ccm_dff` command use `-o %outfile` to write the results of the diff to `%outfile` rather than to standard output. (`%outfile` is the file that contains the results of the two diff'd files.)

For information about how to set options in the system or personal `ini` file, see Setting options in the system or personal ini file.

For information about how to set object type options, see Setting object type attribute options.

For information about how to set options using the set command, see Setting options using the ccm set command.

cli.text_editor

Set Option: System or personal `ini` file, or `ccm set` command

Specifies the text editor used to modify an object's source. The `cli` prefix indicates that the default is for command line use. The user interface uses this variable to determine which tool to use to edit a text attribute. Be sure to include the full path name to the program, or the directory must be included in your path.

The default GUI and CLI text editor is `Notepad` on Windows and `vi` on UNIX.

For information about how to set options in the system or personal `ini` file, see Setting options in the system or personal ini file.

For information about how to set options using the set command, see Setting options using the ccm set command.

cli.text_viewer

Set Option: System or personal `ini` file, or `ccm set` command

Specifies the text editor used to view an object's source. The `cli` prefix indicates that the default is for command line use.

The default CLI text viewer is `Notepad` on Windows and `vi` on UNIX.

For information about how to set options in the system or personal `ini` file, see Setting options in the system or personal ini file.

For information about how to set options using the set command, see Setting options using the ccm set command.

conflict_exclude_rules

Set Option: Model object attribute

This attribute will cause the new conflicts to be excluded, depending on attribute values of the conflicting objects. The following syntaxes are supported:

Syntax	Description
`attrname=attrvalue`	Excludes any conflict where the object's `attrname` attribute value matches `attrvalue`.
`attrname!=attrvalue`	Excludes any conflicts where the object's `attrname` attribute value does not match `attrvalue`.
`EXISTS(attrname)`	Excludes any conflicts where the object has an attribute named `attrname`.
`NOT_EXISTS(attrname)`	Excludes any conflicts where the object does not have an attribute named `attrname`.
`MATCHING(attrname)`	Excludes any conflicts where the object's `attrname` attribute value matches that of the project.
`NOT_MATCHING(attrname)`	Excludes any conflicts where the object's `attrname` attribute value does not match that of the self version.

The default value of `conflict_exclude_rules` is unset.

Additional Information:

- The ! and != rules support values of type string and boolean.
- The values specified in the rules cannot contain the new line character.
- None of the rules should contain character sequences = or !=, except as delimiters for the equal/not-equal rules.
- Any lines that the parser does not understand will be ignored.
- Quotes are not needed around attribute names or string values, and should not be used. If present, they will be considered as literals, in other words, part of the name or value.
- The value of the `conflict_exclude_rules` attribute is cached in the model code, so if the rules are changed, user with active sessions will need to restart their sessions to get the new value.
- This attribute must be set manually, or through a model install. No customization interface is provided.

For information about how to set model attribute options, see Setting model object attribute options.

conflict_parameters

Set Option: Model object attribute.

Specifies which types of conflicts users in this database will see when they show conflicts for a project. The default value of the attribute lists each type of conflict and whether or not conflicts of that type are displayed when you request to see conflicts for a project.

The default editor displays the attribute settings, which contain one conflict setting per line. The line has the following format: `conflict_number: TRUE|FALSE`. Lines that begin with the pound character (#) are treated as comments.

The conflict default values for this option are:

```
# No task associated with object
1: TRUE
# Multiple tasks associated with object
2: FALSE
# Implicitly included object
3: FALSE
# Object included by use operation?
4: TRUE
# Object implicitly required but before baseline
5: FALSE
# Object implicitly required but not included - newer
6: TRUE
# Object implicitly required but not included - parallel
7: TRUE
# Object explicitly specified but before baseline
8: FALSE
# Object explicitly specified but not included - newer
9: TRUE
# Object explicitly specified but not included - parallel
10: TRUE
# Object explicitly specified but no versions of object in project
11: FALSE
# Object implicitly required but no versions of object in project
12: FALSE
# Task implicitly included
13: TRUE
# Task implicitly required but not included
14: TRUE
# Task explicitly specified but not included
15: TRUE
# Task explicitly specified but none of its associated objects
# in project
16: FALSE
# Excluded task explicitly included
17:  TRUE
# Excluded task implicitly included
18: TRUE
```

```
# Completed fix task not included
19: TRUE
# Assigned fix task not included
20: FALSE
# Task fixed by this task not included
21: FALSE
# Implicit task from explicit object
22: TRUE
# Implicitly required by multiple tasks - newer
23: TRUE
# Implicitly required by multiple tasks - parallel
24: TRUE
```

See Conflict detection for more information about conflicts, including a description of each conflict.

For information about how to set model attribute options, see Setting model object attribute options.

copy_db_always

Set Option: System or personal `ini` file

On Windows, forces a database copy when set to `TRUE`.

On UNIX, forces a database copy on `ccm start -rc` when set to `TRUE`.

The default for `copy_db_always` is unset, which causes a database copy to occur only when the `_timetag` file has been touched.

For information about how to set options in the system or personal `ini` file, see Setting options in the system or personal ini file.

date_modified

Set Option: Model object attribute

Creates a keyword to indicate the last time the file was modified. The default model does not include a `date_modified` keyword. You can create this keyword and set its value to the current time by adding the line `date_modified current_time` to the selection set. This will indicate when the file was checked in, if the keyword is expanded on check in.

For information about how to set model attribute options, see Setting model object attribute options.

dcm_broadcast_dbid

Set Option: Model object attribute

Creates a database ID used as an identifier in order to receive transfer packages for the correct database. If `dcm_broadcast_dbid` is set to a non-blank string, DCM initialize automatically creates a DCM database definition for the broadcast database using the value of that attribute as the DCM database identifier. If `dcm_broadcast_dbid` is set to a non-blank string, then DCM receives DCM transfer packages that were generated for a matching DCM broadcast database ID.

The default setting is `TRUE`.

For information about how to set model attribute options, see Setting model object attribute options.

dcm_log_enabled

Set Option: Model object attribute

Specifies that a `dcm_log` attribute be created and updated after the import phase of a DCM receive. This will show each object that DCM tells import or XML import to process. Each line will be of the form:

`<action> from transfer set "<tset>" from database <dbid> on <date>`

where `<action>` is one of the following:

```
created
updated (<A|R|AR[I])
```

`A` is attrs eligible for update
`R` is relations eligible for update
`I` is `image handling`

`<tset>` is the transfer set name
`<dbid>` is the database ID

The `dcm_log` attribute will be excluded by export and XML export, and ignored by import and XML import. It will not be copied on checkout.

The default setting is `FALSE`. This option is intended for use by Support personnel in order to assist customers in debugging DCM issues. This option should remain disabled unless such debugging is required.

For information about how to set model attribute options, see Setting model object attribute options.

dcm_time_sync_tolerance

Set Option: Model object attribute

Controls the amount of time (in seconds) subtracted from the server's current time to compensate for different time settings between machines accessing the database. For additional information about synchronizing servers used in DCM transfers, see "Synchronize Engines and Servers" in *Rational Synergy Distributed*.

The default setting is `60` seconds.

For information about how to set model attribute options, see Setting model object attribute options.

default_task_query

Set Option: System or personal `ini` file

Specifies a user-defined query that can be used to specify a folder's query. You can modify this default query by using query values, which are described in Query expressions.

When you specify the `default_task_query` option in the `ini` file, the `ccm folder` command's `task_scope` option contains a `user_defined` value. Selecting the `User Defined` value causes the query specified by the `default_task_query` option to be part of the folder's query.

When the `default_task_query` is specified in the `ini` file, the `-task_scope` option of the `ccm folder` command supports the `user_defined` value. Using `task_scope user_defined` with the `ccm folder` command will use the query specified by the `default_task_query` option as part of the folder's query.

For information about how to set options in the system or personal `ini` file, see Setting options in the system or personal ini file.

default_version

Set Option: Model object attribute

Specifies the default string for the first version of an object. Use this option to specify an alternative first version string, such as `0001`.

The default for `default_version` is `1`.

For information about how to set model attribute options, see Setting model object attribute options.

engine_host

Set Option: System or personal `ini` file

Specifies the machine on which the Rational Synergy engine runs.

The default for `engine_host` is unset.

This option is not used by the Rational Synergy GUI.

For information about how to set options in the system or personal `ini` file, see Setting options in the system or personal ini file.

expand_on_checkin

Set Option: Set on the cvtype

Allows the database administrator to add an attribute to any cvtype, which will cause keywords to expand on the specified `cvtype`. Keyword expansion normally occurs at check out, but with this option will occur at checkin.

To force keyword expansion at checkin for objects of a certain type, add the `expand_on_checkin` attribute to any type. For example, to enable it for all text type objects, add the `expand_on_checkin` attribute to the `ascii` cvtype; this value will be inherited by all objects in the ascii hierarchy.

The default for `expand_on_checkin` is unset. This option is boolean, so it must be set to either `TRUE` or `FALSE`.

An example of how to enable this for the `ascii` type is

```
$ ccm set role ccm_admin
$ ccm query -t cvtype -n ascii
$ ccm attr -c expand_on_checkin -t boolean -v TRUE @1
```

html_browser

Set Option: System or personal `ini` file

Specifies the HTML browser used to view Rational Synergy HTML help. The value must be the fully qualified path to the executable; for example:

```
html_browser = /usr/local/bin/netscape
```

The default on Windows is `ccm_exec`; on UNIX the default is `ccm_browser`.

This option is not used by the Rational Synergy GUI.

For information about how to set options in the system or personal `ini` file, see Setting options in the system or personal ini file.

html_default_file

Set Option: System or personal `ini` file

This setting operates only on UNIX operating systems.

Specifies an alternative default HTML help file. The default HTML page is displayed when you invoke the HTML help.

The default is `ccm.htm`.

For information about how to set options in the system or personal `ini` file, see Setting options in the system or personal ini file.

html_location

Set Option: System or personal `ini` file

Specifies an alternative Rational Synergy HTML help file location. The value can be an Internet location.

The default is `$CCM_HOME/help`.

For information about how to set options in the system or personal `ini` file, see Setting options in the system or personal ini file.

include_required_tasks

Set Option: Model object attribute

Specifies that when a task is added to Added Tasks of a project grouping, the required tasks on which the task depends are computed and added as well.

The default is unset.

For information about how to set model attribute options, see Setting model object attribute options.

initial_role

Set Option: System or personal `ini` file

Specifies the role with which you start the Rational Synergy CLI. The role and user name determine the access that you have to the objects in the system.

Besides setting your role in your `ini` file, you can change roles by executing the `ccm set` command (described in role). When using the `ccm set` command, the variable name is `role`.

The default for `intial_role` is unset.

> **Note** To change roles successfully using the `ccm set` command, be sure you have privileges for the role you are changing to; otherwise the command will fail.

For information about how to set options in the system or personal `ini` file, see Setting options in the system or personal ini file.

initials

Set Option: System or personal `ini` file

Specifies the default next version for a project or product object to be the initials you specify. The next version will default to `initials` when you check out a project or product object if the new project or purpose has a private purpose. If this option is unset, the default next version for private projects and products is your user name. (The default next version is numeric for all other purposes, regardless of the `initials` option setting.)

To change the default next version to use your initials, enter the following in the system or personal initialization file:

`initials=`*`your_initials`*

For example:

```
initials=leb
```

The default for `initials` is unset.

For information about how to set options in the system or personal `ini` file, see Setting options in the system or personal ini file.

mail_cmd

Set Option: System or personal `ini` file

Rational Synergy uses a default mail tool for DCM e-mail notification. If you want to use your own mailer instead of Rational Synergy's mailer, enter the following line in the `[Options]` section in your `ini` file.

`mail_cmd = `*`user-defined_mail_command`*

The syntax for *user-defined_mail_command* depends on the mailer you want to use. However, your mailer typically will require recipients, subject, and content options and arguments. For example, the following is a `mail_cmd` definition for `ccmail`:

```
mail_cmd = C:\ccmail\mailer.exe -r %recipients -s %subject -f %content
```

The `%recipients`, `%subject`, and `%content` arguments are expanded automatically by Rational Synergy, which uses the information you supply in your dialogs.

For information about how to set options in the system or personal `ini` file, see Setting options in the system or personal ini file.

migrate_check_required_task

Set Option: System or personal `ini` file

When this option is set to `TRUE`, the migrate operation enforces task requirements for creating and checking out new versions, and for checking in checked-out versions' predecessors. The task requirement is defined in each type's Require Task At option.

The default is `TRUE`.

For information about how to set options in the system or personal `ini` file, see Setting options in the system or personal ini file.

migrate_default_arch_state

Set Option: System or personal `ini` file

Specifies the default initial state used when migrating archive files.

The default is `integrate`.

For information about how to set options in the system or personal `ini` file, see Setting options in the system or personal ini file.

migrate_default_state

Set Option: System or personal `ini` file

Specifies the default initial state used when migrating files.

The default is `integrate`.

For information about how to set options in the system or personal `ini` file, see Setting options in the system or personal ini file.

migrate_default_type

Set Option: System or personal `ini` file

Sets the default file type to be used when migrating files into Rational Synergy. The default types cannot be specified to `project` or `dir` when these types need special handling.

> **Note** The default type will be used for create and reconcile operations, as well as migrate.

The default is `ascii`.

> **Note** Rational does not support ISO-Latin-1(ISO 8859-1) character set.

For information about how to set options in the system or personal `ini` file, see Setting options in the system or personal ini file.

multiple_local_proj_instances

Set Option: Model object attribute

Sets the behavior on project creation. Normally, if a project (any version) is created and another instance of that project exists and that project is local to the database, the create will fail. Multiple instances of the same project cannot be created locally. However, if a non-local project has been received from another database, this does not prevent a local project from being created with the same name.

If this attribute is set to `TRUE`, multiple local project instances are allowed. In a non-DCM initialized database, whenever a project is created, it will start with an instance of 1; if that instance already exists, the next available instance number is used. If the user specifies a project without an instance number, then 1 is assumed by default.

The default is `FALSE`.

For information about how to set model attribute options, see Setting model object attribute options.

parallel_exclude_rules

Set Option: Model object attribute

Contains a set of rules defining which version will be excluded form parallel notification.

The following syntaxes are supported:

Syntax	Description
`attrname=attrvalue`	Excludes any parallel CVs whose `attrname` attribute's value matches `attrvalue`. Example: `status=rejected`
`attrname!=attrvalue`	Excludes any parallel CVs whose `attrname` attribute's value does not match `attrvalue`. Example: `is_product!=TRUE`
`EXISTS(attrname)`	Excludes any parallel CVs that have an attribute named `attrname`. Example: `EXISTS(is_product)`

Syntax	Description
EXCLUDE_NON_LEAF_NODES	Excludes any non-leaf nodes on parallel versions.
NOT_EXISTS(attrname)	Excludes any parallel CVs that don't have an attribute named attrname. Example: NOT_EXISTS(is_product)
MATCHING(attrname)	Excludes any parallel versions for which the attrname attribute's value matches that of the self version. Example: MATCHING(release)
NOT_MATCHING(attrname)	Excludes any parallel versions for which the attrname attribute's doesn't match that of the self version. Example: NOT_MATCHING(release)

The default value of this attribute is:

```
status=rejected
is_product=TRUE
EXCLUDE_NON_LEAF_NODES
NOT_MATCHING(release)
```

Additional Information:

- The ! and != rules support values of type string and boolean.
- The values specified in the rules cannot contain the new line character.
- None of the rules should contain character sequences = or !=, except as delimiters for the equal/not-equal rules.
- Any lines that the parser does not understand will be ignored.
- Quotes are not needed around attribute names or string values, and should not be used. If present, they will be considered as literals (in other words, part of the name or value).
- This attribute must be set manually, or through a model install. No customization interface is provided.
- Some customers may want to add the following rules to the default value, to exclude parallel variant branches from notification:

```
NOT_MATCHING(release)
NOT_MATCHING(platform)
```

- The rule `MATCHING(owner)` should not be used. This rule will not work for checkout, because it uses the version you are deriving *from* to detect parallels.

For information about how to set model attribute options, see Setting model object attribute options.

proj_idx_wa_cache

Set Option: System or personal `ini` file

Controls the size of the second work area path cache. The default value is `2500`. Users can increase this value to improve the performance of file accesses in very large projects.

For information about how to set options in the system or personal `ini` file, see Setting options in the system or personal ini file.

project_subdir_template

Set Option: Model object attribute or ccm set command or **Options** dialog box

Enables you to change the default template defining the project-specific directory of your work area path. Setting this option affects the work area path for projects created after the setting has been saved; it does not change the work area path of existing projects.

When changing the value of this option from the command line, you set the `project_subdir_template` variable. This automatically sets the option for the given platform where your interface is running - either UNIX or Windows. The setting is persistent and applies to all sessions on all clients for the given user in the given database.

When changing the model-wide default setting, you need to append `_unix` or `_windows` to the name of the attribute, to indicate whether the template applies to Windows or UNIX work areas. For example, to set a model-wide template for UNIX work areas, you would create an attribute called `project_subdir_template_unix`.

This option is available in the **Options** dialog box as **Add project-specific directory**. This setting is also persistent and applies to all sessions on all clients for the given user in the given database.

The following keywords are valid:

`%project_name` replaces `%project_name` with the new project name.
`%project_version` replaces `%project_version` with the new version.
`%release` replaces `%release` with the new release value.
`%platform` replaces `%platform` with the new platform name.
`%delimiter` replaces `%delimiter` with the new delimiter.

The default is `%project_name%delimiter%project_version`.

If you need to change the non-project-specific portion of the work area path, see wa_path_template.

For information about how to set model attribute options, see Setting model object attribute options.

For information about how to set options using the set command, see Setting options using the ccm set command.

range_for_keyword_expand

Set Option: System or personal `ini` file

Establishes how many characters in a file will be scanned for keywords when an object is created or derived, starting from the beginning of the file.

When you check out a file, it scans the file and replaces keywords with values. If you have a large file with keywords defined in all parts of the file, the amount of time to scan the entire file can cause the create or check-out operation to be very slow.

The default number is `2048`, which refers to the maximum number of characters that will be scanned for keywords. (If your file is set up for 80 characters per line, the default setting will allow at least the first 33 lines per file.)

The default setting works well if you have all of your keywords in the header area. If the keywords are spread throughout your file, you will need to reset this preference so that the keyword expansion can be done throughout the file.

For information about how to set options in the system or personal `ini` file, see Setting options in the system or personal ini file.

reconcile.control_files_below_new_project

Set Option: System or personal `ini` file

Specifies whether uncontrolled files are added to new projects derived from directories during the reconcile operation.

The default is FALSE.

For information about how to set options in the system or personal ini file, see Setting options in the system or personal ini file.

reconcile.save_uncontrolled

Set Option: System or personal ini file

Specifies whether uncontrolled files that are removed from the work area due to conflict resolution should be saved in the work area wastebasket. Setting the option to TRUE will store an uncontrolled file in the work area wastebasket if the file is removed from the work area by an Update Work Area from Database resolution.

The default is FALSE.

For information about how to set options in the system or personal ini file, see Setting options in the system or personal ini file.

reconf_consider_all_cands

Set Option: Model object attribute.

Specifies that a directory be populated with the best match when there are no candidates in a project's update (reconfigure) properties. If this attribute does not exist or if the value if FALSE, the directory entries are left empty when there are no candidates in the project's update properties.

The default is FALSE.

If you want your model to match Synergy release 4.5 or earlier, you must set this option to TRUE.

For information about how to set model attribute options, see Setting model object attribute options.

reconf_release_score

Set Option: Model object attribute.

Specifies that release scoring be used to select object versions whose release best matches the project's release. Release scoring is not used by default for task-based update, but may be considered if you develop parallel releases and one release includes the other release's changes. As there are definite caveats to using this option, it should be used only after serious consideration. For additional information, see the Rational Information Center at http://publib.boulder.ibm.com/infocenter/rsdp/v1r0m0/index.jsp/

The default is FALSE.

For information about how to set model attribute options, see Setting model object attribute options.

reconf_stop_on_fail

Set Option: System or personal `ini` file

Stops the update (reconfigure) process when an individual operation fails. When set to `TRUE`, update stops if an individual operation within the update fails. When set to `FALSE`, the update process continues if an individual operation fails, allowing you to find all errors at one time.

The default is `TRUE`.

For information about how to set options in the system or personal `ini` file, see Setting options in the system or personal ini file.

reconfigure_parallel_check

Set Option: System or personal `ini` file

Indicates whether parallel version notification is given on update (reconfigure).

This option can have the value `FALSE`, `TRUE`, or `FULL`. If set to `FALSE` or not specified, no parallel detection is done. If set to `TRUE`, parallel detection is done only among the candidates chosen by the update selection rules. This setting shows parallel versions specified by the saved baselines and tasks. If set to `FULL`, parallel detection is done among all version of the selected object.

The default is `FALSE` (no notification).

For information about how to set options in the system or personal `ini` file, see Setting options in the system or personal ini file.

reconfigure_using_tasks

Set Option: Model object attribute

Indicates whether you are using task-based Rational Synergy to update projects. This setting applies to your entire database.

The default is `TRUE`.

For information about how to set model attribute options, see Setting model object attribute options.

release_phase_list

Set Option: Model object attribute

Defines the various phases of development or deployment of a release. This feature allows you to track the status of a release during the development process. You can customize this list to match the development phases of your products, or use the default list. The default phase list contains the following phases: `New, Requirements Definition, Function Definition, Implementation, Validation,` and `Released.`

The model attribute is formatted with each entry on a separate line. The default value when a release is created is the first value in the list.

For information about how to set model attribute options, see Setting model object attribute options.

replace_subproj

Set Option: System or personal ini file

Indicates whether the update (reconfigure) operation replaces subprojects as default behavior.

This option can have the value `TRUE` (replace subprojects during update), or `FALSE` (do not replace subprojects).

The default is `TRUE` (that is, replace subprojects).

This option is used by the CLI and Synergy Classic, but is not used by the Rational Synergy GUI.

There is another option in the **Options** dialog box for replacing subprojects in the GUI. That option only applies to the GUI.

For information about how to set options in the system or personal `ini` file, see Setting options in the system or personal ini file.

required_attributes

Set Option: Object type attribute.

Specifies whether users are required to fill in certain fields prior to transitioning an object of the given type to a static state. If one of the required fields is missing or contains an illegal value, the object will not transition to the static state.

The contents of this attribute should be the name of the required attributes, one per line. For example, if you want release, task description and priority to be required fields on a task, you should specify:

```
release
task_description
priority
```

The task will not be able to be completed unless the attributes specified each have a valid value.

The default is an empty string.

For information about how to set object type attribute options, see Setting object type attribute options.

restrict_reconf_setting

Set Option: Model object attribute.

Specifies whether developers can change their projects' update (reconfigure) properties from "object status" to "tasks," and vice-versa. This option also controls whether the update properties can be set at the time a project is created.

> **Note** With a setting of `FALSE`, each user can change his update properties whenever he wants, which could result in unexpected build results. If set to `FALSE`, be sure teams agree on the type of update properties to use.

By default this option is set to `TRUE`, and developers are restricted from changing the update properties setting. This option is a model object attribute. You must be working as a build manager or be in the *ccm_admin* role to set or change this option.

For information about how to set model attribute options, see Setting model object attribute options.

role

Set Option: System or personal `ini` file, or `ccm set` command

Specifies the default role for using the Rational Synergy CLI.

To change the default role in your initialization file, use the initial_role option.

To change roles successfully using the `ccm set` command, be sure you have privileges for the role you are changing to; otherwise, the command will fail.

The default is *developer*.

This option does not affect the Rational Synergy GUI.

For information about how to set options in the system or personal `ini` file, see Setting options in the system or personal ini file.

For information about how to set options using the set command, see Setting options using the ccm set command.

role_definitions

Set Option: Model object attribute

This attribute on the model object specifies what privileges are available to users in the various roles.

You would also modify this attribute to:

- Change the default roles that are allowed to modify a process rule (an update or reconfigure template), by adding or removing the privilege PRIVILEGE_MANAGE_PROCESS_RULES from a given role.
- Add a new role to manage releases by adding the privilege PRIVILEGE_MANAGE_RELEASES to the new role.

- Change the default roles that are allowed to create and assign tasks, by adding or removing the privilege PRIVILEGE_CREATE_AND_ASSIGN_TASKS from a given role.
- Change the default roles that are allowed to assign DCM tasks, by adding or removing the privilege PRIVILEGE_ASSIGN_FOREIGN_TASKS from a given role.

After modifying this attribute, you need to restart your sessions.

For information about how to set model attribute options, see Setting model object attribute options.

save_to_wastebasket

Set Option: System or personal `ini` file

Causes any uncontrolled file in your work area that needs to be removed to be moved to a wastebasket directory. If the `update_db_from_workarea` option is `TRUE`, files involved in collisions with controlled files are copied to the database, not to the wastebasket.

The default is `TRUE`.

For information about how to set options in the system or personal `ini` file, see Setting options in the system or personal ini file.

shared_project_directory_checkin

Set Option: System or personal `ini` file

Causes non-writable directories in shared projects to be checked in to the *integrate* state automatically when objects are added to or deleted from such directories.

The default is `TRUE`.

For more information about shared projects, see Shared projects.

For information about how to set options in the system or personal `ini` file, see Setting options in the system or personal ini file.

start_day_of_week

Set Option: Model object attribute

Specifies the start day of the week to be used when calculating queries that use relative time keywords: `%this_week_begin`, `%this_week_end`, `%last_week_begin`, and `%last_week_end`. Valid entries are integers from 0 - 6, with 0 being Sunday, 1 being Monday, etc.

The default is `0`.

For information about how to set model attribute options, see Setting model object attribute options.

sync_output

Set Option: System or personal `ini` file, or `ccm set` command

If you want to display sync status messages from the command line, you do not need to do anything; the default is to display the messages. If you do not want the messages to be displayed, you will need to set the `sync_output` option in one of the following two ways.

For information about how to set options in the system or personal `ini` file, see Setting options in the system or personal ini file.

For information about how to set options using the set command, see Setting options using the ccm set command.

system_filename_filters

Set Option: Model object attribute.

Specifies the database default file patterns to be ignored when users sync the work area. This is set by the CM administrator (the *ccm_admin* role).

If certain files are being ignored in the work area based on their extension, and they should be included, the solution is to remove the extension from the `system_filename_filters` attribute.

The table below lists the default filters.

Default filters	
*.APS	*.BAK
*.BSC	*.class
*.CLW	*.ENC
*.EXP	*.IDB
*.ILK	*.INCR
*.MD#	*.MD~
*.MD%	*.NCB
*.OBJ	*.OPT
*.PCH	*.PDB
*.PLG	*.PROJDATA
*.RES	*.SBR
*.SUO	*.TLH

Default filters	
*.TLI	*.TMP
*.USER	*.WBK
_vti*	_vti**
~*	*~
Copy of *	New Folder
timestamp.inf	

For information about how to set model attribute options, see Setting model object attribute options.

update_on_checkin_if_equal

Set Option: System or personal `ini` file

When you use some editors or you perform scripted check-outs and check-ins, the timestamps on the database and work area versions of a file can appear to be identical. When set to `TRUE`, the `update_on_checkin_if_equal` option forces Rational Synergy to copy such files from the work area to the database, even though their timestamps indicate that they are not newer than their database versions.

The default is `FALSE`.

For information about how to set options in the system or personal `ini` file, see Setting options in the system or personal ini file.

verbosity

Set Option: System or personal `ini` file

Specifies the default verbosity for messages output from the `ccm update` command. A level of 5 or greater causes additional information to be displayed by the update operation. The model your database uses also can use the verbosity level.

The default is `0` (zero), the lowest setting.

The **Options** dialog box also has a verbosity setting, which has the same effect as this setting.

For information about how to set options in the system or personal `ini` file, see Setting options in the system or personal ini file.

wastebasket

Set Option: System or personal `ini` file

Use the `wastebasket` option to specify the location of your wastebasket directory.

%database becomes the name of the database for which the wastebasket is being used. (The wastebasket directory is hidden.)

%database and %user are keywords you can use to specify the wastebasket path to create directory names that differ for each user and/or database using the same template. These keywords are replaced at startup. If the directory does not exist, Rational Synergy creates it.

The default path resides in your home directory and is:

```
Windows:  HOME\%user\ccm_wa\.moved\%database
UNIX:     $HOME/%user_name/ccm_wa/.moved/%database
```

where %*user* replaces %*user* with your user name.

For information about how to set options in the system or personal ini file, see Setting options in the system or personal ini file.

wa_path_cache_size

Set Option: System or personal ini file

Controls the size of the work area path cache. The default value is 500. Users can increase this value to improve the performance of file accesses in large projects.

For information about how to set options in the system or personal ini file, see Setting options in the system or personal ini file.

wa_path_template

Set Option: Model object attribute

Enables you to change the default template defining the non-project-specific directory of your work area path. Setting this option affects the work area path for projects created after the setting has been saved; it does not change the work area path of existing projects.

When changing the value of this option from the command line, you set the wa_path_template variable. This automatically sets the option for the given platform where your interface is running - either UNIX or Windows. The setting is persistent and applies to all sessions on all clients for the given user in the given database.

When changing the model-wide default setting, you need to append _unix or _windows to the name of the attribute, to indicate whether the template applies to Windows or UNIX work areas. For example, to set a model-wide template for UNIX work areas, you would create an attribute called wa_path_template_unix.

Set the following path using the ccm set command.

```
ccm set wa_path_template %home\%database\location
```

The following keywords are valid:

`%database` replaces `%database` with the new database name.
`%user` replaces `%user` with your user name.
`%owner` replaces `%owner` with the project owner's name.
`%home` replaces `%home` with your home directory.

The Windows default is `%home\ccm_wa\%database`, where `%home` is your home directory (wherever you designated that to be in the **Startup Info** dialog's **Home Directory** text box).

The UNIX default is `%home/ccm_wa/%database`, where. `%home` is your UNIX home directory.

This option is available in the **Options** dialog box as **Set default path for all work areas**. This setting is also persistent and applies to all sessions on all clients for the given user in the given database.

If you need to change the project-specific portion of the work area path, see project_subdir_template.

For information about how to set model attribute options, see Setting model object attribute options.

For information about how to set options using the set command, see Setting options using the ccm set command.

Initialization file - Windows

Run Rational Synergy from the PC server

Typically, the `ccm.ini` file is located in two places: the system file is located in the interface's installation area, *`CCM_HOME`*`\etc`, and a personal file usually exists in each user's Windows `Documents and Settings` directory. If you do not have a `ccm.ini` file in your Windows `Documents and Settings` directory, you can copy the system file, and then modify it with the options you want to set.

The personal `ccm.ini` file overrides the system file.

You can edit your `ccm.ini` file by using any text editor.

Run Rational Synergy on your PC

If you are running Rational Synergy from an installation on your PC, the `ccm.ini` file is in *`CCM_HOME`*`\etc`. You can modify this file directly since it will affect your machine only.

Initialization file - UNIX

The default `ccm.ini` file is located in `$CCM_HOME/etc/ccm.ini`. You can copy the default file to your home directory, and then modify it for your use. Once you do this, the settings in your `.ccm.ini` file will override those of the system `.ccm.ini` file. Even if you haven't changed any settings, if you don't already have a personal `.ccm.ini` file, it is created automatically when you exit from your session.

You can create and edit your personal `.ccm.ini` file by using any text editor. The `.ccm.ini` file must reside in your home directory. Options that are preceded with the word *Motif* affect only the graphical user interface. When no interface is specified, the option applies to all interfaces, as appropriate.

Startup file

Startup file commands are stored in a Rational Synergy file called `ccminit`. You can use these commands to do the following:

- Define commands
- Define aliases
- Run commands

You can change these commands:

a. for an installation area,

b. for a particular database, or

c. for your personal session.

The Windows startup files are read and executed in the following order:

- *CCM_HOME*\etc\ccminit
- *database_directory*\lib\ccminit
- *user's_home_directory*\ccminit

The UNIX startup files are read and executed in the following order:

- *CCM_HOME*/etc/ccminit
- *database_directory*/lib/ccminit
- *user's_home_directory*/.ccminit

If all three startup files exist, Rational Synergy will read and execute the commands in all three; each additional startup file provides extra commands to execute at startup.

Rational Synergy reads the installation area settings first, then the database-specific settings, and then the personal settings. The commands in the last file read override those read from the previous files.

> **Note** Regardless of which startup file you change, you must restart your session to make the new command(s) available.

Installation area setting

Setting commands in this file affects all users of an installation area. These commands are set in the system startup file (`ccminit`).

The system startup file resides in the directory:

Windows: *CCM_HOME*\etc
UNIX: $*CCM_HOME*/etc

Database-specific setting

Setting commands in this file affects all users of a specific database. These commands are set in the database startup file (`ccminit`).

The database startup file resides in the directory:

Windows: *database_directory*`\lib`
UNIX: *database_directory*`/lib`

Personal setting

Setting commands in this file affects only your own session. You can set these defaults in your personal startup file (`ccminit` on Windows, `.ccminit` on UNIX).

If you do not already have a personal startup file, you will need to create one in your home directory.

Example

Assume that you want to set an alias permanently for your session only, but that you do not have a `ccminit` file in your home directory.

1. Create a startup file in your home directory.
2. Add your alias to the startup file.

   For example, your startup file might contain the following lines:

   ```
   alias my_tasks "task -query -task_scope all_my_assigned"
   alias cidt "task -checkin default"
   alias exit stop
   ```

   Note You can add as many aliases as you want to the startup file, but each alias should be on its own line.

   Note Do not include the leading `ccm` for Rational Synergy commands. For example, the command to set an `alias` is `ccm alias`. Notice that in the last line of the example above, the command contains `alias` only.

3. Save your changes, and then exit from the startup file.
4. If you have a session running, exit from Rational Synergy, and then restart a session.

   The alias is available for your use.

GUI settings

You can change the settings for the Rational Synergy graphical user interface by using the **Options** dialog box.

Environment variables

You can define the following variables to affect the way Rational Synergy runs. The following table shows environment variables that you can set.

Environment variable	Required	Use
AUTOMOUNT_FIX (UNIX only)	No	Used to determine the portion of path names that should be stripped to support automounter usage. Not needed if the /tmp_mnt prefix is used by the automounter. This variable is also used by SunOS automounter patch. For more information, refer to the *Rational Synergy Administration Guide for UNIX*.
CCM_ADDR	No	Specifies the remote function call (RFC) address (*host*:*socket*) for the Rational Synergy interface.
CCM_ENGLOG	No	Used to redirect the engine log. If not set, the ccm_eng.log file in your Windows installation directory or UNIX home directory is used. The engine log file must be visible and writable by *ccm_root*.
CCM_HOME	No	Specifies the Rational Synergy installation directory, typically C:\Program Files\IBM\Rational\Synergy 7.1 for a Windows client, and /usr/local/ccm on UNIX.
CCM_PAGER (UNIX only)	No	Specifies the name of an executable used to display a file or report output using ccm monitor on UNIX. This takes preference over PAGER, if set.
CCM_UILOG	No	Used to redirect the user interface log. If not set, the ccm_ui.log in your installation directory on Windows or home directory on UNIX is used.
DISPLAY (UNIX only)	Yes	Name of X display server; for example, unix:0.0.
HOME (UNIX only)	Yes	Specifies your UNIX home directory.
LD_LIBRARY_PATH (Sun only)	Yes	Specifies a list of directories used to search for dynamic object libraries, for example: /usr/lib/X11:/usr/openwin/lib

Environment variable	Required	Use
PAGER (UNIX only)	No	Specifies the name of an executable used to display a file or report output using `ccm monitor` on UNIX.
PATH	Yes	Specifies a list of directories used to search for executables.
PRINT_EDIT_CMD	No	If this variable is set to a value, the model-defined editor method that is being used is displayed whenever the editor command is executed.
PRINT_TOOL_CMD	No	If this variable is set to any value at all, the model-defined tool method (for example, debug, print, or execute) is displayed whenever the tool command is executed.
RECONF_TIME	No	Times and displays the execution of an update (reconfigure) command.
SHELL (UNIX only)	Yes	Specifies the name of a UNIX command interpreter shell to invoke for subprocesses, for example, `/bin/csh`.
TERM (UNIX only)	Yes	Specifies the type of terminal from which UNIX commands are entered, for example, `xterm`.
UIDPATH (UNIX only)	No	Specifies a list of directories used to search for files.
USER (UNIX only)	Yes	Specifies your user name.

Note Rational Synergy uses other variables starting with `CCM_` or `AC_` for internal diagnostic purposes. **Do not** set any such variables, nor `INFORMIXDIR`, `INFORMIXSERVER`, or `ONCONFIG` variables, unless you are told to do so by Support personnel.

Setting model object attribute options

You can set model-wide attributes on model objects; these settings affect all users of databases into which that model has been installed. You must be in the *ccm_admin* role to change model attribute objects.

The first example shows how to create the attribute, which is required in some cases. After creating the attribute, you can then set it. The second example shows how to modify an attribute that is already set.

Substitute the appropriate option name and syntax for the option you are changing.

Create an attribute

This example uses the `allow_deliminiter_in_name` attribute, which specifies whether the version delimiter is allows in an object name. By default, this attribute does not exist, which means the delimiter is not allowed.

To set this option the first time, you must create the attribute as follows:

1. Set your role to *ccm_admin*.

   ```
   ccm set role ccm_admin
   ```

2. Query for the model object in the Rational Synergy database.

   ```
   ccm query -t model -n base
   ```

3. Create the attribute.

   ```
   ccm attr -c allow_delimiter_in_name -t boolean @1
   ```

4. Set the value.

   ```
   ccm attr -m allow_delimiter_in_name -v TRUE @1
   ```

5. Restart your session. (All users of this database must restart their sessions.)

Modify an attribute

This example uses the `wa_path_template_unix` attribute, which specifies the default non-project-specific work area directory. By default, this directory is `%home/ccm_wa/%database`.

To modify this attribute, do the following:

1. Set your role to *ccm_admin*.

   ```
   ccm set role ccm_admin
   ```

2. Query for the model object in the Rational Synergy database.

   ```
   ccm query -t model -n base
   ```

3. Specify the new path.

```
ccm attr -m wa_path_template_unix @1 -v "%home/workareas/%database"
```

4. View the new contents of the attribute.

   ```
   $ ccm attr -show wa_path_template_unix @1
   ```

5. Change back to your previous role.

   ```
   ccm set role previous_role
   ```

6. Restart your session. (All users of this database must restart their sessions.)

Creating a list box for a new attribute

You can create list boxes for newly created attributes. The syntax for creating new list boxes is as follows:

attr_name:attr_type[:[label][:#textlines]] |
attr_name:attr_type[:[label]:[#textlines]:values_ref]

where `values_ref` is defined in a new values definition entry, in a separate attribute.

Each `values_ref` values definition entry must be defined in a separate text attribute on an object or type called `info_attrs.values_ref`, where `values_ref` is the name of the list of values referred to in the `info_attrs` definition. This allows the list of values to be easily populated from external tools.

A `values_ref` must be a legal attribute name because it becomes part of the name of an attribute. A `values_ref` attribute name is limited to 21 characters because the limit on the length of an attribute name is 32 characters and 11 characters are used by the string, `info_attrs`.

The contents of an `info_attrs.values_ref` attribute must be a newline-separated list of possible values.

A value in a value list may be any ASCII string with embedded white space allowed, but no leading or trailing white space (since such white space will be considered part of the newline delimiter).

Example:

Suppose you want to add a custom attribute to the task type called `approval_level` and the possible values for this attribute are from the following list:

" new

" pending

" approved level 1

" approved level 2

You could create an entry in the info_attrs attribute on the task type as follows:

approval_level:string:Approval Level::approval_values

You could then create an attribute on the task type called `info_attrs.approval_level` with the following contents:

new

pending

approved level 1

approved level 2

Setting object type attribute options

You can set object type attributes on objects; these settings affect all users of databases having that object type. You must be in the *ccm_admin* role to change object types.

Substitute the appropriate option name and syntax for the option you are changing.

1. Set your role to *ccm_admin*.

   ```
   ccm set role ccm_admin
   ```

2. Query for the type whose setting you want to change.

   ```
   ccm query -t cvtype -n misc
   ```

3. Modify the attribute.

   ```
   ccm attr -m required_attributes @1
   ```

   This will bring up an editor on the attribute. Make your changes, and then save the value.

4. Change back to your previous role.

   ```
   ccm set role previous_role
   ```

5. Restart your session. (All users of this database must restart their sessions.)

Setting options in the system or personal ini file

Some options can be set by changing your personal `ini` file or using the `ccm set` command. Changing the option using your personal `ini` file will cause the change to be in effect every time you start a session. Changing the option using the `ccm set` command makes the change at the run-time level (that is, you do not need to restart a session for the change to take effect).

The following examples change the `cli.text_editor` option, the first in your Windows `ccm.ini` file, and the second, in your UNIX `.ccm.ini` file. Substitute the appropriate option name and syntax for the option you are changing.

- To specify `Notepad` as the default editor, your `ccm.ini` file should contain the following setting:

  ```
  cli.text_editor=notepad %filename
  ```

- To specify `vi` as the editor, your `.ccm.ini` file should contain the following settings:

  ```
  cli.text_editor="vi %filename"
  ```

Setting options using the ccm set command

Some options can be set by changing your personal `ini` file or using the `ccm set` command. Changing the option using your personal `ini` file will cause the change to be in effect every time you start a session. Changing the option using the `ccm set` command makes the change at the run- time level (that is, you do not need to restart a session for the change to take effect).

The following example changes the `wa_path_template` option.

Substitute the appropriate option name and syntax for the option you are changing.

Enter the following to change the work are path template:

```
ccm set wa_path_template %home/workareas/%database
```

Commands

alias command

Synopsis

```
ccm alias [alias_name ["string"]]
```

Description and uses

The `alias` command enables you to create another name for a command. Use this command in the following way.

- With no arguments, `alias` lists all defined aliases.
- With a single argument, `alias` prints the value of *`name`*.
- With two arguments, `alias` sets the new alias *`name`* to the value *`string`* or changes the value of the existing alias *`name`* to *`string`*.

Setting the `alias` command applies the alias to the current session only. To learn how to set the alias permanently, see Startup file.

Options and arguments

`name`

Specifies the name of the alias.

`string`

Specifies the command to be substituted by the alias.

An alias is a direct textual substitution, so *`string`* can be a command name, a full command, or part of a command. If *`string`* is a command with arguments, enclose the entire command in quotes.

Examples

- List all defined aliases.

  ```
  ccm alias
  ```

- Create an alias to check out a file with a new version.

  ```
  ccm alias getf "checkout -t"
  ```

  When you use the new alias, it will be in the following form:

  ```
  ccm getf myversion foo.c
  ```

- Change the value of an alias.

  ```
  ccm alias alias_name "new alias value"
  ```

For example, assume you have an alias, `my_query`, defined to query for objects. Now you want to change the value of `my_query` to query for tasks. You would change the `my_query` alias's value to query for tasks by running the `alias` command.

Related topics

- unalias command

attribute command

Synopsis

Copy Attribute

```
ccm attr|attribute -cp|-copy [attr_name[:attr_name...]
                   [-append] from_file_spec to_file_spec
                   [to_file_spec...]
ccm attr|attribute -cp|-copy attr_name[:attr_name...]
                   [-append] [-subproj] [-suball]
                   -p|-project from_project_spec
                   to_project_spec [to_project_spec...]
```

Create Attribute

```
ccm attr|attribute -c|-create attr_name [-f|-force]
                   -t|-type type [-v|-value value]
                   file_spec [file_spec...]
ccm attr|attribute -c|-create attr_name [-f|-force]
                   -t|-type type [-v|-value value]
                   -p|-project project_spec [project_spec...]
```

Delete Attribute

```
ccm attr|attribute -d|-delete attr_name file_spec [file_spec...]
ccm attr|attribute -d|-delete attr_name
                   -p|-project project_spec [project_spec...]
```

Modify Text Attributes

```
ccm attr|attribute -m|-modify attr_name [-v|-value value]
                   file_spec [file_spec...]
ccm attr|attribute -m|-modify attr_name [-v|-value value]
                   -p|-project project_spec [project_spec...]
```

Modify Non-Text Attributes

```
ccm attr|attribute -m|-modify attr_name -v|-value value
                   file_spec [file_spec...]
ccm attr|attribute -m|-modify attr_name -v|-value value
                   -p|-project project_spec [project_spec...]
```

Show Attributes

```
ccm attr|attribute -s|-show attr_name file_spec [file_spec...]
ccm attr|attribute -s|-show attr_name
                   -p|-project project_spec [project_spec...]
```

List Attributes

```
ccm attr|attribute -l|-la|-li file_spec [file_spec...]
ccm attr|attribute -l|-la|-li
                   -p|-project project_spec [project_spec...]
```

Description and uses

Use the `attribute` command to manipulate the Rational Synergy attributes associated with objects. If the attribute is a `text` attribute, `-v` is optional.

Options and arguments

`-append`

Appends the specified attribute value(s) to the specified object. If you do not use this option, any existing values for the specified attributes are overwritten by the new values.

`-cp|-copy attr_name[:attr_name...]`

Copies an attribute or set of attributes to a selected set of object or project versions in a single operation. You can use the colon as the separator character if you want to specify more than one attribute name.

`-c|-create attr_name`

Creates an attribute.

`-d|-delete attr_name`

Deletes an attribute.

`file_spec`

Specifies the file or directory whose attribute will be changed (that is, modified, deleted, etc.).

`-f|-force`

You must use the `-type` option with the `-force` option.

Checks whether the attribute to be created exists and has the same type, and then causes one of the following to occur:

- If the attribute to be created exists and has the same type, the attribute's value is changed (if you use the `-value` option).

- If the attribute does not exist, the new attribute is created.
- If an attribute with the same name exists, but has a different type, the operation fails.

The difference between `ccm attr -c attr_name -t type` and `ccm attr -c attr_name -f -t type` is that the command without the -`force` option fails if the attribute already exists.

from_file_spec to_file_spec

When used with -`cp`, this option specifies that *from_file_spec* is the file from which the attribute is copied and *to_file_spec* is the file to which the attribute is copied.

`-l`

Lists all local attributes.

`-la`

Lists all attributes.

`-li`

Lists the inherited attributes.

`-m|-modify` *attr_name*

Modifies an attribute. If you do not specify the `-v` option, an editor is invoked on the attribute.

`-p|-project` *from_project_spec to_project_spec*

When used with -`cp`, this option specifies that *from_project_spec* is the project from which the attribute is copied and *to_project_spec* is the project to which the attribute is copied. If -`subproj` or -`suball` is used, the project is applied to *to_proj_spec*.

`-p|-project` *project_spec*

Specifies the project whose attribute will be changed (that is, deleted, copied, etc.).

`-s|-show` *attr_name*

Shows the value of an attribute.

`-suball`

Recursively copies the specified attribute(s) to subproject objects and all members of the specified project. This option applies to *`to_proj_spec`* and requires the `-p` option.

`-subproj`

Recursively copies the specified attribute or set of attributes to the subproject objects in the specified project. This option applies to *`to_proj_spec`* and requires the `-p` option.

`-t|-type` *`type`*

Specifies the type of the attribute. Use this option only when you create attributes. Valid built-in values include the following:

- string (used for single-line `ascii` attributes)
- boolean
- text (used for multi-line `ascii` attributes)

`-v|-value` *`value`*

Specifies the value of the attribute.

Examples

- Create a string attribute named `new_attr` for the `driver.c` object.

  ```
  ccm attr -c new_attr -type string driver.c
  ```

- Show the value of the `comment` attribute for the `driver.c` object.

  ```
  ccm attr -s comment driver.c
  ```

- Change the `release` attribute of `foo.c` to `4.2_int`.

  ```
  ccm attr -m release -v 4.2_int foo.c
  ```

- Copy the `version` attribute from a project, `attr_test-1`, to its subprojects.

  ```
  ccm attr -copy version -project attr_test-1 -subproj attr_test-1
  Copying attrs: version
  from: attr_test-1.
      to: attr_test2-1:project:1.
      to: attr_test3-1:project:1.
      to: attr_test4-1:project:1.
  Attribute Copy completed with 0 errors
  ```

baseline command

Synopses

Compare Two Baselines

```
ccm baseline -compare baseline_spec1 baseline_spec2
           [-tasks] [-objects] [-projects] [-change_requests]
```

Create or Preview a Baseline

```
ccm baseline -c|-create
           [-rehearse]
           -p|-project project_spec -p|-project [project_spec. . .] |
           -bl|-baseline baseline_spec [-baseline baseline_spec…] |
         -pg|-project_grouping project_grouping_spec [project_grouping_spec.
. . ]
           [-r|-release release]
           [-purpose purpose_spec]
           [-d|-description "baseline_description"]
           [-subprojects|-no_subprojects|-all_subprojects]
           [-vt|-version_template version_template]
           [s|-state state_name]
           [-b|-build build_string]
           [baseline_name]
```

Delete a Baseline

```
ccm baseline -delete baseline_spec
            [-wp|-with_projects_and_products]
            [-np|-no_projects_and_products]
```

List Baselines

```
ccm baseline -l|-list [-release release] [-purpose purpose_spec]
           [-f|-format "format_string"] [-ns|-no_sort] [-u|-un_numbered]
```

Mark a Baseline for Deletion

```
ccm baseline -mfd|-mark_for_deletion
            baseline_spec
```

Modify a Baseline

```
ccm baseline -modify baseline_spec
           [-n|-name name]
           [-b|-build build]
           [-v|-versions [-vt|-version_template version_template]
             [-skip_nonvisible_projects]]
           baseline_spec
```

Publish a Baseline

```
ccm baseline -publish baseline_spec
```

Release a Baseline

```
ccm baseline -rb|-release_baseline
           [-comment comment_string]
           baseline_spec
```

Restore a Deleted Baseline

```
ccm baseline -undelete
           baseline_spec
```

Show a Baseline

```
ccm baseline -sh|-show
           i|info|information|
           proj|project|projects|
           obj|objs|objects|
           t|task|tasks|
           cr|change_request|change_requests

(fcr|fully_included_change_request|fully_included_change_requests)|
           (pcr|partially_included_change_request|
           partially_included_change_requests))
           [-f|-format "format_string"]
           [-ns|-no_sort]
           [-u|-un_numbered]
           baseline_spec
ccm baseline -sh|-show
           ((r|release)|(p|purpose)|(o|owner)|(desc|description)|(b|build))
           baseline_spec
```

Description and uses

A baseline is a set of projects and tasks used to represent your data at a specific point in time. A baseline has many uses. When you perform an update, Rational Synergy uses a baseline as a starting point to look for new changes. You can also compare two baselines

to see what changes have been made relative to a particular build. If you use Rational Change, you can use baselines to generate change request reports.

Typically, a build manager creates a baseline; a developer doesn't need to create a baseline because he doesn't make his builds available to other users.

You might find it useful to create a baseline as soon as you perform a build. You can create a baseline and make it available to the test group without making it available to all developers. Making the baseline as soon as you build saves a representation of the build in Rational Synergy in case it's needed later to create a fix for that particular build.

Creating a baseline for each `Integration Testing` and `System Testing` build enables testers and developers to refer back to the set of changes that were used to create the build. Typically, you'll create a baseline for all projects in the same release and purpose. For example, you would create a baseline for each `Integration Testing` build using all `Integration Testing` projects for that release.

> **Note** When you create a baseline, you'll specify a list of projects to be included in the baseline. Be sure to include all related projects in your baseline so that you have a complete set for reference

Baselines can be used by process rules to define the baseline for the projects that use that template. For example, a build manager may create a baseline named `Integration Build 20040913` containing static projects `toolkit-int_20040913`, `calculator-int_20040913`, etc. The numeric designation is the date (*yyyymmdd*) the baseline was created.

A process rule can specify that its projects use a particular baseline; the projects that reference that process rule use the baseline to identify which baseline project to use when updating. For example, if the `Integration Testing` process rule for the current release specified that the `Integration Build 20040913` baseline should be used, a developer's `calculator-bob` project would select `calculator-20040913` as its baseline project.

Using baselines has the following benefits:

- Build managers have a lightweight way to save a set of projects that were successfully built and tested.
- Process rules are more flexible; they can specify a particular baseline or the latest baseline with certain characteristics, enabling build managers to control the team's process more precisely. If problems were discovered in a newer baseline, the build manager can reset the team's baseline to a previous, successfully-built baseline.
- The update operation uses the baseline to streamline which tasks are evaluated, thereby improving update performance. Only those tasks on top of the baseline are considered when computing update candidates. When a baseline is created, the set of tasks is taken from either the project grouping or from the projects themselves for the projects that update manually. In addition, the tasks from the project grouping's baseline are added to the new baseline, unless the release is different.

- Team members can compare baselines to identify which tasks were introduced in the latest baseline, or identify whether a baseline includes a particular task. This is useful for testers who need to know what features to test, as well as whether to expect a known problem to be fixed in a particular build.
- Specify that a project should be updated to match the latest successful build.

How is a new baseline created?

Both *prep*-state projects and static projects may be added to a new baseline. However, if a *prep* project is added to a baseline, the actual project is not added. Instead, a copy of the project is created and added to the baseline and checked in. Prep projects and their work areas are preserved as is so as not to cause unnecessary rebuilds. Moreover, new versions are checked out and checked in for all non-static products that are members of the *prep* project. Other than that, the new project has the same members as the *prep* project. The new projects and products are checked in to the `member_status` that is associated with the baseline's purpose. If this `member_status` is not a valid state, the projects and products are checked in to the *integrate* state.

For example, a baseline that has the `Integration Testing` purpose has projects and products that are in the *integrate* state.

If a prep project contains any non-static members that are not projects or products, you cannot add it to a baseline. Before you can add such a project to a baseline, you must check in its non-static members. In addition, you cannot add to a baseline any project whose update properties include a task that is not complete.

A new project's or new product's version is created based on the *prep* project's version, the date, and if necessary to make the version unique, an incremental number that is appended. For example, if project `ccm_gui-sol_int` is saved as part of a baseline, the new baseline project becomes something like `ccm_gui-sol_int_20040709`. If it is not possible to append an underscore, the date, and an incremental number to the existing version string (and also stay within the limit of 32 characters), then just the date and the number are used.

After a baseline is created, the history view links are changed so that it appears that existing *prep* projects are checked out from the new baseline projects. In addition, project histories are updated to make it look as though existing prep products are checked out from the products that are created for the baseline projects.

New projects that are created as part of a baseline do not have work areas. If you want the projects to have work areas, you must enable work area maintenance after the baseline has been created. When a project that has a visible work area is added to a baseline, it is checked for work area conflicts. If any non-resolvable conflicts are found, the create baseline operation will fail. To resolve this issue, you must reconcile the project.

If a project with a non-visible work area is added to a baseline, the latest-built product may not have been copied to the database. In such a case, the baseline will contain what is in the database, not what is in the non-visible work area. To avoid this problem, the build manager must make sure that changes to all non-visible work areas of projects that are

added to a baseline have been synchronized to the database. This must be done before adding such projects to a baseline.

Use the `baseline` command to:

- Create a baseline from an existing prep hierarchy or set of hierarchies.
- Save a baseline instead of manually populating the Tested Tasks folder in order to publish the latest tested changes to developers.
- Show information or associated projects, objects, and tasks for a specific baseline.
- List baselines.
- Modify or rename a baseline
- Release a baseline or compare two baselines.
- Delete an existing baseline, or mark a baseline for deletion.
- Restore a deleted baseline.

You must be working as a build manager to create or release a baseline. You must be working as in the *ccm_admin* role to delete a baseline or modify the build of a released baseline. Any user can show, compare, or list baselines.

Options and arguments

`-all_subprojects`

When this option is used with `-create`, specifies that entire project hierarchies be added to a baseline. The default behavior is -subprojects.

`-baseline` *`baseline_spec`*

If this option is used with `-create` and one or more *`baseline_specs`* are specified, the projects in the specified existing baselines are added to the new baseline.

Note that the `–subprojects`, `-no_subprojects`, and `–all subprojects` options will affect which subprojects are also added.

By default, if the `–baseline` option is used, but the `–project` option is not, subprojects are not included. However, if the `–project` and `–baseline` options are used together, then the `–subprojects` default implied by `–project` overrides the `–no_subprojects` default implied by `–baseline`.

baseline_name

The *baseline_name* is the name that is assigned to the baseline. When you create a baseline, you can assign any legal object version name to the baseline.

If you do not specify a *baseline_name* when you carry out the `ccm baseline -create` or `-modify` command, a unique name is automatically assigned to the baseline. This default name is a name that is in the form `yyyymmdd`. If needed, the default name is followed by an underscore and an incremental number to make it unique. For example, the first baseline created on April 1, 2002 has a default name of `20020401`. The second such baseline created on the same day has a default name of `20020401_1`.

baseline_spec

The *baseline_spec* allows the *baseline_name* or selection set reference form to be used where a baseline name is allowed. The entire selection set reference, @, can be used. For more information, see Baseline specification.

When you release a baseline, you can specify a *baseline_spec* that includes a leading DCM database ID (dbid) and a DCM delimeter (for example, `J#;` where `J` is the dbid and `#` is the DCM delimeter).

`-build` *build_string*

If this option is used with `-create`, the create operation uses the *build_string* for the new baseline.

When used with `-modify`, allows the build string on a baseline to be changed. A build can be changed by a user working as a build manager, unless the baseline is in the *released* state. A baseline that has been released can be modified only by a user working in the *ccm_admin* role.

`-cr|-change_request|-change_requests`

If this option is used with `-show`, the show operation shows the partially and fully included change requests in the baseline. The default format is: `%displayname: %problem_synopsis`.

If this option is used with `-compare`, the compare baseline operation shows differences in change requests between the two baselines. For example, when comparing two baselines B1 and B2, the comparison output includes change requests

fully included in both B1 and B2, change requests partially included in both B1 and B2, and change requests that are fully or partially included in one project or the other.

`-comment` *`string`*

If a comment is specified, it is appended to the comment on all baseline projects and their members. This occurs when the baseline is released and those projects and members are checked in.

`-compare`

If this option is used with `-baseline`, compares two baselines that have the specified *`baseline_specs`*. Shows the differences between the properties of two baselines, compares different versions of the same project, shows projects that were either added to or removed from the baseline, and shows differences in change requests between the projects. It also shows the differences in tasks of the two baselines.

`-create`

Creates a new baseline. If the baseline name that you specify is already in use by a baseline that is local to the database, the command will fail. Moreover, if you specify an invalid or non-active release, the command will fail.

The baseline you create includes all static projects in the hierarchies that you specify by the *`project_spec`*. It includes all static projects and copies of *prep* projects. If any projects in the hierarchy do not match the release or purpose that you specify, the command will succeed; however, a warning message will be displayed. The Selection set reference form or the full selection set (`@`) can be used as the *`project_spec`*. To learn more about *`project_spec`*, see Project specification.

You can add *prep*-state projects to baselines that you create.

The project hierarchies that you specify can contain modifiable products. If any other modifiable members are not products or projects, the create baseline operation will fail. If you do not specify the release and purpose, the release and purpose of the first specified project is used.

`-delete`

Deletes the baseline that has the *`baseline_spec`* you specify. You must be in the *ccm_admin* role to use this option. If `-wp` is specified, the baseline is deleted with projects and products. If `-np` is specified, only the baseline is deleted. The default behavior is to delete the baseline and the projects and products that did not exist in a static state before the baseline was created.

You cannot delete a baseline if any non-static project uses that baseline, or if a process rule uses the baseline. In addition, if you try to delete a baseline and its checked-in projects and products, and one or more of its associated projects or products is a member of a project that is not part of the baseline, the delete baseline operation will succeed; however, those projects or products will not be deleted. Furthermore, if one or more projects that are in the baseline are members of another baseline, or are baseline projects, the delete operation will be successful, but those projects will not be deleted.

The `-delete` option works with multiple baselines, including a single item selection set reference, such as `@[0-9]+`, or the entire selection set reference.

`-description "`*`baseline_description`*`"`

Optionally, provides a detailed description of the baseline. There is no limit on its length, and no restriction on its contents. The *`baseline_description`* must be enclosed in double quotes if it contains one or more spaces.

`-f|-format "`*`format_string`*`"`

Specifies the command's output format. The default format depends on the options that you use with `-format` (for example, `-list` or `-show`) and those options' keyword arguments. To learn more about the default output formats, see the descriptions of the options that you can use with `-format`.

The format can contain a combination of text and keywords. Keywords are replaced by specific data about each object. For example, the keyword `%owner` is replaced with `sue` if information about an object owned by user `sue` is displayed.

`-list`

Lists baselines. If `-release` or `-purpose` is specified, only those baselines that match the release or purpose are listed.

`-mfd|mark_for_deletion`

Marks the baseline for deletion. A baseline marked for deletion can be deleted later with save offline and delete (SOAD), or manually by a user working in the *ccm_admin* role. Note that a baseline does not have to be in the *deleted_baseline* state to be deleted by a user in the *ccm_admin* role.

`-modify`

Allows various attributes of the baseline to be changed. If the version template is updated, it is updated after the name and build attributes are modified, as the template may be expanded differently if the name or build is changed. You must be working as a build manager to make modifications to a baseline.

The `name` and `build` keywords in a template expand to the newly specified name and build if they were also specified.

If a work area is visible, but cannot be updated for other reasons, such as lack of proper file permissions or lack of disk space, the operation fails and the command returns an error code of 1, regardless of the setting of the `-skip_nonvisible_projects` option. However, the operation continues, even on failure, reporting all failures to update work area at the end.

`-name`

When used with `-modify`, allows the baseline to be renamed. A baseline can be renamed by a user working as a build manager, unless the baseline is in the *released* state. A baseline that has been released can be modified only by a user in the *ccm_admin* role.

`-np|-no_projects_and_products`

If this option is used with `-delete`, causes the baseline to be deleted, but does not delete projects that are associated with the baseline, and the products in those projects.

`-no_subprojects`

When used with `-create`, specifies that no subprojects be included in the baseline.

`-ns|-no_sort`

Specifies that the command's output will not be sorted.

`-objects`

If this option is used with `-compare`, the compare baseline operation shows common objects or objects that were added or removed between the two baselines.

-project

If this option is used with -create and a *project_spec* is specified, it indicates the projects is to be added to the baseline. By default, when a project is added, its entire hierarchy is also added. The -no_subprojects option can be used to override this.

-projects

If this option is used with -compare, common projects and projects unique to each baseline are listed.

-project_grouping *project_grouping_spec*

If this option is used with -create and one or more *project_grouping_specs* are specified, the projects in the specified project groupings are added to the new baseline. By default, when a project grouping is added, only those projects in the project grouping are added; subprojects that are not part of the project grouping are not added. The –all_subprojects option can be used to override this.

Note that the –subprojects, -no_subprojects, and –all subprojects will affect which subprojects are also added.

By default, if the –project_grouping option is used, but the –project option is not, subprojects are not included. However, if the –project and –project_grouping options are used together, then the –subprojects default implied by –project overrides the –no_subprojects default implied by –project_grouping.

project_spec

Each *project_spec* that is specified represents a project that is to be included in the baseline. To learn more about *project_specs*, see Project specification.

-publish *baseline_spec*

Transitions baselines in the test_baseline state to the published_baseline state. You must be working as a build manager to complete this transition.

-purpose *purpose_spec*

Specifies the baseline purpose. If not specified, the purpose defaults to the purpose of the first project specified.

`-rehearse`

Lists the projects and products that will make up the baseline.

If any version conflicts are found, either when creating a baseline or when using the `-rehearse` option, you will see a warning, listing all the product and project versions that are in conflict, either because an existing version already exists, or because the resulting version would not be a legal version string.

`-rb|-release_baseline`

Releases the baseline that has the *`baseline_name`* you specify. Also checks in all of the baseline's projects and their members to the *released* state.

`-release` *`release`*

Specifies the release value of the baseline. When creating a baseline, any active release value may be used. If not specified, the release defaults to the release of the first project specified.

`-show`

Shows the release, purpose, and project information that is associated with the *`baseline_name`* you specify. Also shows the release and purpose for each project that is in the baseline.

`i|info|information`

If specified, the following information is displayed.

Name
Description
Release
Purpose
Released
Projects
Build

`r|release`

If specified, the baseline's release value is displayed.

`p|purpose`

If specified, the baseline's purpose is displayed.

`o|owner`

If specified, the name of the baseline's owner is displayed.

`desc|description`

If specified, the baseline's description is displayed.

`projects`

If specified, all projects that are included in the baseline are displayed. The default format is:

```
%displayname %status %owner %release %create_time
```

The default format may be overridden by using the `-format` option.

`objects`

If specified, all objects that are included in the baseline are displayed. The default format is:

```
%displayname %status %owner %release %create_time
```

The default format may be overridden by using the `-format` option.

`tasks`

If specified, all tasks that are included in the baseline are displayed. The default format is:

```
%displayname %release %owner %create_time
```

The default format may be overridden by using the `-format` option.

`cr|change_requests`

If specified, all change requests included in the baseline are displayed.

`fcr|fully_included_change_request|fully_included_change_requests`

If specified, only fully included change requests are displayed.

`pcr|partially_included_change_request|partially_included_change_requests`

If specified, only partially included change requests are displayed.

`-skip_nonvisible_projects`

When this option is used with `-modify`, it specifies that projects without a visible work area will not be changed.

For each work area that cannot be updated because it is not visible, a warning is displayed; the operation continues and is successful if there are no other problems and `-skip_nonvisible_projects` was used. If `-skip_nonvisible_projects` was used, an error is returned, but the operation continues and does not clean up. All errors are reported at the end. The message indicates whether the failure was because the work area was not visible or because it could not be modified

`-state`

Specifies the state of the baseline when it is created. When creating a baseline, valid states are *test_baseline*, *published_baseline*, and *released*. The default state for a baseline is *test_baseline*. Developers can see the baseline in this state and can use it manually. They won't get is automatically as the latest baseline. SQA can use it for testing. Once it passes testing, the build manager must transition the test baseline to *published_baseline* to make it available for developers to use.

Creating a baseline in the *released* state is equivalent to creating one in the *published_baseline* state, and then releasing it.

`-subprojects`

When this option is used with `-create`, specifies that project hierarchies be added to a baseline. This includes all *prep* subprojects, and nonmodifiable subprojects will be included if the component part of the release of that nonmodifiable subproject matches the release of the baseline. This must be an exact match of the component name. A release without a component name can only match another release without a component name. This is the default behavior if no options are specified.

`-tasks`

If this option is used with `-compare`, common tasks and tasks unique to each baseline are listed.

`-u|-un_numbered`

Suppresses automatic numbering of the command's output (that is, the output is un-numbered).

-undelete

Restores a baseline in the *deleted_baseline* state to the state it was in before it was deleted. If the baseline is not in the *deleted_baseline* state, no change occurs.

-ver|-versions

Specifies that the version of the projects and products are to be modified.

version_template

A *version_template* is any string, with optional keywords, which can be of the form %*keyword* or %{*keyword*}. The keyword can be any Rational Synergy attribute or a built-in keyword.

When an attribute is expanded, the corresponding attribute value from the prep project or product being examined is used. If no attribute or built-in keyword is found for a specified keyword name, the empty string is used to replace the keyword.

-vt|-version_template *version_template*

When used with -create, all new project and product versions that are checked in during the command use the *version_template* for their versions.

When used with -modify, the versions of the project and product versions that became static when the baseline was created are updated to match *version_template*. However, projects that existed in a static state before the baseline was created are not reversioned. For example, if a CM/6.3 SP3 baseline was created with 20 existing static projects from the CM/6.3 SP2 baseline and 5 new projects from the CM/6.3 SP3, only the 5 new projects will be reversioned.

If the instantiated version_template for any project or product in the baseline contains characters that are not allowed in a version string, then those characters are replaced with the default version string replacement character. This is specified in the ccm.ini file, with the option baseline_template_repl_char. This character default is the underscore character (_). For example, if %platform is part of a project version template, and the prep project has a platform of SPARC-solaris, then the version string contains the string SPARC_solaris. Or, if %release is part of a product version template, and the prep product has a release of CM/7.1, then the version string contains the string CM_7.1.

If the instantiated *version_template* for any project or product in the baseline is already in use for another version of that project or product, then the version is made

unique by appending the underscore character (_) and the first integer, starting with `1`, that will make the version unique. If this causes the version string to be too long, then a version based on the current date is used for that project or product, and a warning is given.

If `-version_template` is not specified, then the default (i.e., saved) template is used. For more information, see Version template specification.

The work area is updated if the work area template for the project includes the version. If a work area cannot be updated because it is not visible, and `-skip_nonvisible_projects` is not used, the operation continues and all errors are reported. If the work area is visible, but cannot be updated for other reasons, such as lack of proper file permissions or lack of disk space, the operation continues and all failures are reported.

`-wp|-with_projects_and_products`

If this option is used with `-delete`, it causes the projects that are associated with a baseline, and the products in those projects, to also be deleted.

Examples

- Display a list of baselines for release `2.2` and purpose `Integration Testing`.

  ```
  ccm baseline -list -release 2.2 -purpose "Integration Testing"
  ```

- Compare the projects that are in a baseline named `20020401_1` and a baseline named `20020401_2`.

  ```
  ccm baseline -compare 20020401_1 20020401_2 -projects
  ```

- Create a baseline named `Build_1234_int` for Release 2.0, for the purpose of Integration Testing, that includes a project named `proj1-sqa_3` and its subprojects.

  ```
  ccm baseline -c Build_1234_int -d "Integration build 1234" -r 2.0 -
purpose "Integration Testing" -projects proj1-sqa_3 -subprojects
  ```

Related topics

- update_properties command
- process_rule command

bom command

Synopsis

```
ccm bom file_spec [file_spec...]
```

Description and uses

The `bom` command enables you to display the Bill-of-Materials to standard output for one or more specified objects.

You also can view a BOM from a product object's Properties dialog.

Any user can execute this command.

Options and arguments

`file_spec`

Specifies the name of the file for which the Bill-of-Materials is displayed. *`file_spec`* must be a controlled product.

Example

- View a Bill-of-Materials.

  ```
  ccm bom file_spec
  ```

candidates command

Synopsis

`ccm cand|candidates [-recommend] file_spec`

Description and uses

The `candidates` command lists all versions of an object that are eligible for selection when you perform a use or update operation in a directory entry. An object is a candidate for use if the name, type, and object instance attribute values of the object match those of the directory entry.

The output shows each object version's name, version, state, owner, project in which it was created, instance, and associated task number.

Options and arguments

file_spec

Specifies the name of the object or directory entry for which the candidate versions are listed.

`-recommend`

Adds an asterisk (*) to the listed version that would be selected by Rational Synergy based on its selection rules.

Example

- List the versions of `Xincls.h` that can be members of the current project in the directory, and recommend the version to use.

```
ccm cand Xincls.h -recommend
1) Xincls.h-1 integrate chrisb incl projX 1 5
2) Xincls.h-2 integrate chrisb incl projX 1 12
3) Xincls.h-3 integrate terri  incl projX 1 13
4) Xincls.h-4 integrate terri  incl projX 1 15 *
```

Related topics

- update command
- use command

cat command

Synopsis

`ccm cat` *`file_spec`* [*`file_spec`*`...`]

Description and uses

The `cat` command displays the source of an object. This command is useful for displaying the contents of an object that is not currently a member of the directory.

Options and arguments

`file_spec`

Specifies the name of the file to be displayed.

Example

Display the second instance of the `foo.c-9` object version, which is a `csrc` object.

```
ccm cat foo.c-9:csrc:2
```

Related topics

- view command
- type command

change_type command

Synopsis

```
ccm change_type file_spec -t|-type new_type -task task_number
```

Description and uses

Changes the type of a specific object. A new version of the object that has the specified type is created. If the specified object is in the *working* state, it is replaced with the new object, and the specified object is deleted from the database. If the object is a member of a project and the `change_type` command is executed within the project, the old object is replaced by the new object in the project. If the parent directory is not modifiable, it is automatically checked out for you. The project must be writable by you. If the object is a member of more than one project, or if the command is not executed within the project where the object is a member, the command fails.

Any user can perform this operation.

Options and arguments

`-type new_type`

Specifies the new type that the object will have.

`file_spec`

Specifies the name of the file whose type you are changing.

`-task task_number`

Associates any newly-created directory and object with the specified task.

If the current (default) task is set and you do not specify a different task, the newly-created directory is associated with the current task automatically.

Example

- Change the type of the file.

  ```
  ccm change_type file_spec -t|-type new_type
  ```

checkin command

Synopsis

```
ccm ci|checkin [-s|-state state]
               [-nc|-nocomment] [-cr|-commentreplace]
               [-c|-comment "string"]
               [-ce|-commentedit]
               [-cf|-commentfile file_path]
               [file_spec [file_spec...]
ccm ci|checkin [-source] [-products] [-projects]
               [-h|-hierarchy] [-task task_number]
               [-s|-state state]
               [-ps|-product_state product_state]
               [-ss|-source_state source_state]
               [-nc|-nocomment] [-cr|-commentreplace]
               [-c|-comment "string"]
               [-ce|-commentedit]
               [-cf|-commentfile file_path]
               -p|-project [project_spec project_spec...]
```

Description and uses

Use the `checkin` command to check in one or more objects and, if necessary, set the next state.

You can check in source (non-product objects), product and project objects, assign task numbers to objects you will check in, and add, modify, or replace a comment for the object you will check in.

> **Note** You should make changes from only one work area, and you should perform your check-ins with that work area visible.

Options and arguments

`-c|-comment "string"`

Enables you to add the *string* as the object's comment. If you are checking in a task, the comment is added to the task.

`-ce|-commentedit`

Brings up an editor to enter the comment.

`-cf|-commentfile` *`file_path`*

Uses the comments from the specified file. If you specify both a comment string and comment file, the comments are merged, with the comment string following the comments from the file.

`-cr|-commentreplace`

Normally, any newly specified comment(s) is appended to an existing comment. Use the `-cr` option to replace an existing comment. You can replace a comment only on writable objects.

`file_spec`

Specifies the file or directory you want to check in.

`-h|-hierarchy`

Applies the check-in scope (for source, products, or projects) to the project hierarchy.

`-nc|-nocomment`

Do not prompt for comments. (Normally, if you do not specify a comment with either the `-c` or `-cf` option and the object does not already have a comment, you are prompted for a comment.)

`-products`

Checks in all products in the current project.

`-p|-project` *`project_spec`* [*`project_spec...`*]

Indicates that the specified object is a project.

`-projects`

Checks in all projects in the top-level project, and then checks in the top-level project.

`-ps|-product_state` *`product_state`*

Use this option with the `-products` option.

Specifies the state of product objects when checking in a product. This applies both to hierarchy and non-hierarchy check-ins (that is, this option does not require the `-s` option).

`-s|-state` *`state`*

Explicitly sets the states of the files or projects to be checked in. If you do not specify a state, the default next state is calculated automatically.

If you enter the `-products` and `-source` options but do not enter the `-p` and `-ps` options, this option (`-state` *`state`*) is used as the next state for products and source objects.

`-source`

Checks in all objects in the current project that are not products or projects.

`-ss|-source_state` *`source_state`*

Enables you to specify the state of non-product objects when checking in a project hierarchy.

Examples

- Check in the current version of `foo.c` with a state of *visible*.

  ```
  ccm checkin -s visible foo.c
  ```

- Check in the directory `utils` without any new comments.

  ```
  ccm ci -nc utils
  ```

- Check in three files (`clear.c`, `concat.c`, and `display.c`).

  ```
  ccm ci -nc clear.c concat.c display.c
  ```

- Check in the `c_includes` symbolic link to the *checkpoint* state (UNIX only).

  ```
  ccm ci -c "let others edit" -state checkpoint c_includes
  ```

- Check in the `projB-3` project.

  ```
  ccm ci -c "configuration sent to customer A" -p projB-3
  ```

- Check in all members of the `tools-5` project, with product members going to the *checkpoint* state, and source (non-product) members going to the *integrate* state.

  ```
  ccm ci -p tools-5 -products -s checkpoint
  ccm ci -p tools-5 -source -ss integrate
  ```

Caveats

To check in a project version to a non-modifiable state, be sure that all members are in a non-modifiable state already because you cannot check in a project to a non-modifiable state if it has modifiable members.

Related topics

- checkout command
- task command

checkout command

Synopsis

```
ccm co|checkout [-task task_number]
                [-t|-to file_spec|version]
                [-c|-comment "string"]
                [-ce|-commentedit]
                [-cf|-commentfile file_path]
                file_spec [file_spec...]
ccm co|checkout [-purpose purpose_spec]
                [-platform platform]
                [-release release|as_is|none]
                [-reconf tasks|os]
                [-subprojects]
                [-versions "old_ver:new_ver,old_ver:new_ver,..."]
                [-t|-to version]
                [-c|-comment "string"]
                [-ce|-commentedit]
                [-cf|-commentfile file_path]
                -p|-project project_spec
```

Check out a project and set work area properties

```
ccm co|checkout check_out_options
                [-cb|-copy_based|-not_copy_based|-ncb  (UNIX only)]
                [-rel|-relative|-nrel|-not_relative]
                [-path|-set|-setpath] absolute_path
                [-mod|modifiable_wa] [-nmod|not_modifiable_wa]
                [-tl|-translate|-ntl|-no_translate]
                [-wa|-maintain_wa|-nwa|-no_wa]
                [-wa_t|-wa_time|-nwat|-no_wa_time]
                [-u|update|-no_u|-no_update]
                -p|-project project_spec
```

Description and uses

Use the `checkout` command to check out objects, and use the copy_project command to check out projects. Note that the `checkout -project` operation is now called the `copy_project` operation in Rational Synergy.

When you check out an object in a non-shared project, its default state is *working*. When you check out a file or directory in a shared project, its default state is *visible* if it is a non-product, and *shared* if it is a product.

If you specify the `-p` option, the specified project (or the entire project hierarchy) is checked out. The `copy_project` command functions the same as the `checkout` command with the `-p` option.

When you check out from a static (non-modifiable) project and do not check out subprojects, and the subprojects have relative work areas, new copies of those subprojects' work areas are created in the appropriate locations within the work area of the project being checked out. This enables developers to reuse static subprojects that have relative work areas.

Static work areas are not maintained and cannot be reconciled with the database; they are ignored during reconcile. Sync'ing a static work area will replace any files that have been modified with files from the database. Checking out a project with a static work area will leave the original work area in place; you must reconcile it to discard or keep changes.

Use the `checkout` command in the two following ways.

Create a Modifiable Version of a File or Directory

When you check out an object, a writable version of the object is placed in the directory. (Use the `ccm dir` (Windows) `or ccm ls` (UNIX) command to verify the object.) When you check out a directory, no visible change is made to the file system. When you use the `-t` option to specify a new version at check out, you can specify the version and change the name of the new object. On UNIX, the checkout of a symbolic link enables you to change the location to which the symbolic link points.

You can use the following forms to check out an object version within a work area:

- project reference form

  `sub_proj\foo.c@my_proj-1` (Windows)
  `sub_proj/foo.c@my_proj-1` (UNIX)

- selection reference form

  `@1`

- object reference form

  `foo.c-1:csrc:1`

- work area reference form

  `c:\users\tom\ccm_wa\main-1\main\src\foo.c` (Windows)
  `/users/tom/ccm_wa/main-1/main/src/foo.c` (UNIX)

From outside of a work area, you can use the project reference form only for objects that are used.(

`sub_proj\foo.c@main-1` (Windows)
`sub_proj/foo.c@main-1` (UNIX)

From outside of a work area, you *must* use one of the following forms if the object is not used anywhere (that is, it is a floating object):

- selection reference form

  `@1`

- object reference form

  `foo.c-1:csrc:1`

Create a Modifiable Version of a Project or Project Hierarchy

By default, when you check out a project, it is created in the database and a work area is created automatically. You can set work area properties at the time you check out the project.

Options and arguments

`-c|-comment "`*`string`*`"`

Specifies the comment string.

`-cb|-copy_based (UNIX only)`

Makes the work area copy based. You can use this option only with the `-p` option.

See work_area command for more information.

`-ce|-commentedit`

Brings up your default editor to enter the comment.

`-cf|-commentfile` *`file_path`*

Uses the comments from the specified file. If you specify both a comment string and comment file, the comments are merged, with the comment string following the comments from the file.

`file_spec`

Specifies the name of the file or directory that you want to check out.

`-mod|-modifiable_wa`

Specifies that the work area can be modified.

`-nmod|-not_modifiable_wa`

Specifies that the work area is not modifiable.

`-no_u|-no_update`

Specifies that the project is not updated when it is copied.

`-not_copy_based|-ncb (UNIX only)`

Makes the work area link-based. You can use this option only with the `-p` option.

See work_area command for more information.

`-ntl|-no_translation`

Indicates that ASCII files should not be translated when they are copied between Windows and UNIX within the project's work area. You can use this option only with the `-p` option.

See work_area command for more information.

`-nrel|-not_relative`

On Windows, makes subprojects' work areas absolute instead of relative to the parent project's work area. This is the default when you first create a project. You can use this option only with the `-p` option.

On UNIX, causes links to be used for subprojects and makes subprojects' work areas absolute instead of relative to the parent project's work area. This is the default when you first create a project. You can use this option only with the `-p` option.

See work_area command for more information.

`-nwa|-no_wa`

Causes the work area not to be maintained (that is, disconnects your work area from the database). You can use this option only with the `-p` option.

See work_area command for more information.

`-nwat|-no_wa_time`

Sets the time stamps of the files in the project's work area to reflect the last modification time stored in Rational Synergy, rather than the time the files were copied into the work area. You can use this option only with the `-p` option.

See work_area command for more information.

`-platform` *`platform`*

Enables you to specify the platform value that is set on the project or project hierarchy that you are checking out. The platform choices are listed in the *`CCM_HOME`*`\etc\om_hosts.cfg` file (Windows) or `$CCM_HOME/etc/om_hosts.cfg` file (UNIX) in your Rational Synergy installation area.

By default, the current platform value for each project in the hierarchy is copied to its new version. You can use this option only with the `-p` option.

`-p|-project` *`project_spec`*

Checks out a project, or if specified with the `-subprojects` option, the entire project hierarchy is checked out.

`-purpose` *`purpose_spec`*

Specifies the purpose to associate with the specified project.

The purpose choices are listed in the Project Purpose Table and include all of the purpose values that are being used in your database.

If you are working as a developer when you perform the check out, this option defaults to `Insulated Development`, but you can specify `Shared`, instead. If you are working as a build manager or in the *ccm_admin* role, `Integration Testing` is the default setting. You can use this option only with the `-p` option.

`-reconf tasks|os`

Enables you to specify whether the project or project hierarchy you are checking out is updated using a baseline and tasks (`tasks`) or the state of the candidate objects (`os`). (The latter is the only method used in releases previous to Synergy 4.2.) You can use this option only with the `-p` option.

`-rel|-relative`

Makes the work area path relative to the parent project's path. You can set this option only if the project is used in one place. After you set it, this project cannot be used as a subproject in any other project. You can use this option only with the `-p` option.

`-release` *`release`*

Enables you to specify the release value that is set on the project or project hierarchy that you are checking out. The release choices include all of the release values that are being used in your database, `as is`, or `none`.

By default, the current release value for each project in the hierarchy is copied to its new version. You can use this option only with the `-p` option.

`-subprojects`

Causes all subprojects in the specified project's hierarchy to be checked out. You can use this option only with the `-p` option.

`-t|-to` *`file_spec|version`*

Enables you to specify the version and/or change the name of the new, non-project object, or specify the version of a new project or project hierarchy.

By default, the `-to` argument is interpreted as a new version. For example, if you execute the following command:

```
ccm co foo.c -to bar
```

the new object version is:

```
foo.c-bar
```

To change the name, you must include the object name and the version in the destination argument. For example, if you execute the following command:

```
ccm co foo.c -to bar.c-1
```

the new object version is:

```
bar.c-1
```

If you are checking out a project, you can specify the version only. If you are checking out a hierarchy of projects, the new version is used for the project as well as its subprojects. Use the `-versions` option to map new versions to old versions of projects in the hierarchy. The `-to` and `-versions` options are mutually exclusive. Also, if you do not specify the `-to` or `-version` option, the default next version is computed automatically using a Rational Synergy built-in algorithm.

If you are checking out a new version of an object that is used in your current project, the newly checked-out version (the "to" version) also will be used in your project.

> **Note** When you check out to a new object name in a non-writable directory, a new directory version is checked out automatically.

If you are in a shared project and your current directory is non-writable, the directory is checked out and associated automatically with the default (or specified) task and is checked in to the *integrate* state. You can disable this feature by setting `shared_project_directory_checkin` to `FALSE` in your initialization file. (See shared_project_directory_checkin.)

`-task` *`task_number`*

Associates the objects being checked out with the specified task.

If the current (default) task is set and you do not specify a different task, the objects you are checking out are associated with the current task automatically.

`-tl|-translate`

Indicates that ASCII files should be translated when they are copied between Windows and UNIX within the project's work area. You can use this option only with the `-p` option.

See work_area command for more information.

`-u|-update`

Specifies to update the project when it is copied. The project is checked out without a work area and is then updated, respecting the project grouping's setting that indicates whether the baseline and tasks should be refreshed. Then the project is synchronized.

`-versions "`*`old_ver`*`:`*`new_ver`*`,`*`old_ver`*`:`*`new_ver`*`,..."`

You can use this option only with the `-p` option.

The default next version is computed using a Rational Synergy built-in algorithm. (In most cases the current version is incremented by "1.") To change the next version, map the old version to the new version using the syntax shown.

If you are checking out a project hierarchy, each mapping applies to all projects in the hierarchy that currently have that value. If *`new_ver`* is `NoCheckOut`, projects with the corresponding *`old_ver`* are not checked out.

Use the `-to` option to specify the same version for all new projects that you are checking out. The `-to` and `-version` options are mutually exclusive.

`-wa|-maintain_wa`

Causes the work area to be maintained (that is, synchronizes the work area and keeps it synchronized). You can use this option only with the `-p` option.

See work_area command for more information.

`-wat|-wa_time`

Sets the timestamps of the files in the project's work area to show the time the files were copied into the work area, rather than the Rational Synergy modification time. You can use this option only with the `-p` option.

See work_area command for more information.

Examples

- Check out version patch1 from version 1 of `foo.c` (version 3 of `foo.c` is in the current directory).

  ```
  ccm co -c "patch1: fix symbol table bug" -to patch1 foo.c-1
  ```

- Check out the `utils\tools` (Windows) or `utils/tools` (UNIX) directory, which currently is at version 4.

  Windows:
  ```
  > ccm co -c "added new files" c:\users\bob\ccm_wa\test_db\projA-3\utils\tools
  ```
  UNIX:
  ```
  $ ccm co -c "added new files" ~/ccm_wa/test_db/projA-3/utils\tools
  ```

- Set the comment and associate a task with the *`object_version(s)`* you are checking out.

  ```
  ccm co -c "comment string" -task task_number object_name1 object_name2
  ```

- Check out a new *working* project hierarchy from an existing project hierarchy. Set the versions of all of the projects to your name.

  ```
  ccm co -p toolkit-int -subprojects -to john
  ```

- Check out a new *prep* project hierarchy for system testing. Set the release and platform values and versions.

  Windows:
  ```
  > ccm co -p tool_top-1.0 -subprojects -release 2.0 -platform win32 -
  purpose sqa -versions
  "1.0:sqa,win16_1.0:win16_sqa,win32_1.0:win32_sqa"
  ```
  UNIX:
  ```
  $ ccm co -p tool_top-1.0 -subprojects -release 2.0 -platform SunOS -
  purpose sqa -versions
  "1.0:sqa,win16_1.0:win16_sqa,win32_1.0:win32_sqa"
  ```

- Modify a top-level project's version and propagate the change to its subproject versions.

  ```
  ccm co -p top_project_spec -subprojects -to version
  ```

Related topics

- checkin command
- copy_project command

checkpoint command

Synopsis

```
ccm ckpt|checkpoint [-t|-to version] [-cr|-commentreplace]
                    [-c|-comment "string"]
                    [-ce|-commentedit]
                    [-cf|-commentfile file_path]
                    [-task task_id]
                    file_spec [file_spec...]
ccm ckpt|checkpoint [-t|-to version][-cr|-commentreplace]
                    [-c|-comment "string"]
                    [-ce|-commentedit]
                    [-cf|-commentfile file_path
                    [-task task_id]
                    -p|-project [project_spec...]
```

Prerequisites

You must own the object to perform a `checkpoint`. Only working objects can be checkpointed.

Description and uses

The `checkpoint` command enables you to save a personal version of an object for your use only, by preserving it in a state that is not modifiable, but that you can delete later when you no longer need it.

When you perform a checkpoint, the current version of the object is moved to the *checkpoint* state and a new version of the object is created. All comments specified on the `checkpoint` command are applied to the checkpointed object.

Options and arguments

`-c|-comment "`*`string`*`"`

Specifies the comment string.

`-ce|-commentedit`

Brings up an editor to enter the comments.

`-cf|-commentfile` *`file_path`*

Takes the comments from the specified file. If you specify both a comment string and comment file, the comments are merged, with the comment string following the comments from the file.

`-cr|-commentreplace`

Normally, the comment specified is appended to any existing comment. However, if you use the `-cr` option, the new comment will replace any existing comment.

`file_spec`

Specifies the name of the file or directory, or project that you want to checkpoint.

`-t|-to` *`version`*

Sets the version of the newly checked-out object. You also can do this by adding the version to the object name.

`-task` *`task_id`*

Specifies the task with which you want your newly checked-out object to be associated.

If you do not specify a task but a current task is set, the newly created object version is associated with the current task. Any task associated with the checkpoint object version remains unchanged.

`-p|-project` *`project_spec`*

Checkpoints a project.

Examples

- Checkpoint the current *working* version of `foo.c`, and add a comment.

```
ccm ckpt -c "Phase 1 works." foo.c
Adding 'release' attribute with value '2.0' to object foo.c-3:csrc:11
Associated object foo.c-3:csrc:11 with task 36
Checkpointed object version: 'foo.c-2:csrc:11'
```

- Checkpoint the current working version of `foo.c`. Add a comment and specify the new *working* object's version to be `joe`.

```
ccm ckpt -c "Trying Jane's algorithm." -t joe foo.c
Adding 'release' attribute with value '2.0' to object foo.c-
joe:csrc:11
Associated object foo.c-joe:csrc:11 with task 36.
Checkpointed object version: 'foo.c-3:csrc:11'
```

Related topics

- collapse command

clean_cache command

Synopsis

```
ccm clean_cache [-t|-type type] [-s|-status status]
                [-c|-cutoff_time time] [-u|-used]
                [-v|-verbose]
```

Description and uses

By default, the `clean_cache` command removes archived cache files of non-modifiable objects that are older than 14 days and not used in work areas.

Cache files for object versions that were not archived are never deleted.

Only users in the *ccm_admin* role can use this command.

On Windows, for more information on deleting cache files, refer to "Delete Cache Files" in the *Rational Synergy Administration Guide for Windows*.

On UNIX, for more information on deleting cache files, refer to "Delete Cache Files" in the *Rational Synergy Administration Guide for UNIX*.

Options and arguments

`-c|-cutoff_time` *`time`*

Specifies that cache files are deleted only for object versions with sources older than *`time`*. The *time* used is the access time of the file, not the modify time. The time default removes files older than 14 days (`-c "-14:0:0:0"`).

`-s|-status` *`status`*

Specifies the status of the object versions for which cache files are deleted. By default, versions in non-modifiable states are selected.

`-t|-type` *`type`*

Specifies the type of the object versions for which cache files are deleted. The default is to delete cache files for all types.

`-u|-used`

Deletes cache files, regardless if they are used in any projects, with or without work areas. By default, cache files that are used in a project with a work area are **not** deleted. On UNIX, the project's work area would then contain symbolic links to files that no longer exist.

> **Note** Do not use this option unless you understand the repercussions of its use.

Examples

- Remove `csrc` cache files that are more than 45 days old.

  ```
  ccm clean_cache -type csrc -cutoff_time "-45:0:0:0"
  ```

clean_up command

Synopsis

```
ccm clean_up -all|-task|-rpt [-used]
             [-q|-quiet | -v|-verbose]
             [-rel|-release release]
             [-user user_name]
```

Description and uses

Use the `clean_up` command to delete unused automatic tasks or to delete process rules from a Rational Synergy database.

You must be working in the *ccm_admin* role to delete process rules. If you attempt to delete process rules while working as the PT administrator, a message appears stating that the process rules will not be deleted.

Also, any user can remove his or her own automatic tasks, but you must be working in the *ccm_admin* role or as the PT administrator to delete automatic tasks owned by another user.

Options and arguments

`-all`

Equivalent to specifying both the `-task` and `-rpt` options. Causes Rational Synergy to delete both unused automatic tasks and unused process rules.

`-q|-quiet`

Minimizes output messages.

`-rel|-release release`

Causes automatic tasks and/or process rules to be removed only for the specified release.

`-rpt`

Causes unused process rules to be deleted.

A process rule is unused if no project is using that process rule to set its update properties.

`-task`

Causes unused automatic tasks to be deleted.

Note If you are working in the *ccm_admin* role and run this command without the `-user` option, all unused tasks in the database will be deleted.

An automatic task is unused if 1) it is not contained in the update properties of any project, and 2) it has no objects associated with it.

When you are not working as the CM or PT administrator, `-task` is the same as `-task -user` *your_user_name*.

`-used`

Causes in-use process rules, as well as unused process rules, to be deleted. Rational Synergy sets update properties to 'manual' on the in-use process rules' projects.

This option has an effect only when used with the `-rpt` option.

`-user` *user_name*

Causes automatic tasks owned by *user_name* to be deleted. You cannot use this option with the `-rpt` option.

`-v|-verbose`

Maximizes output messages.

Examples

- Remove all unused process rules.

  ```
  ccm clean_up -rpt
  ```

- Remove both in-use and unused process rules.

  ```
  ccm clean_up -rpt -used
  ```

- Remove both in-use and unused process rules for release 5.0, and execute the command verbosely.

  ```
  ccm clean_up -rpt -used -rel 5.0 -v
  ```

- Remove all automatic tasks belonging to another user, *user_name*.

  ```
  ccm clean_up -task -user user_name
  ```

collapse command

Synopsis

```
ccm collapse [-from file_spec] file_spec [file_spec...]
ccm collapse -all file_spec [file_spec...]
```

Description and uses

The `collapse` command enables you to delete object versions from the database and adjust history links. You use this command when the object you delete has successors.

All collapsed objects are removed from the database. If you collapse a project, its members are unused and the project is removed.

> **Note** Before you check in a *working* version to a non-modifiable state, be sure to collapse the earlier checkpointed versions you do not want in your database; otherwise, you will only be able to collapse the checkpointed versions while working in the *ccm_admin* role.
>
> **The immediate successor of a checkpointed version must be writable to collapse the predecessor.** For example, assume you have an object named `bufcolor.c` with a version history of: 1 --> 2 --> 3 --> 4, where version 1 is in the *integrate* state, versions 2 and 3 are checkpointed versions, and version 4 is in the *working* state. If you want to collapse version 3, you can do so because version 4 is in the *working* state. If version 4 was in the *integrate* state, you would not be able to collapse version 3.

You can perform a `collapse` command on a specified object except when the object has either of the following characteristics:

- the object is non-modifiable and you are not working in the *ccm_admin* role
- the object is a member of a project

> **Note** Collapsing a version that has multiple successors and predecessors links each of the version's predecessors with each of the version's successors. Collapsing a version that has no predecessors and multiple successors unlinks the successors' histories.

All users can collapse objects to which they have write access.

Users working in the *ccm_admin* role can collapse non-modifiable objects as well.

If a user working in the ccm_admin role performs a `collapse -all` command, a simplified history will result, which generally includes only those object versions that are members of projects. **All** intermediate versions, regardless of their states, are removed.

Options and arguments

`-all`

Indicates that all versions of the file or directory (that is, the entire history) specified by *`file_spec`* will be collapsed.

> **Caution** Working objects will also be collapsed if they are not project members. Use caution when you use the `-all` option.

`file_spec`

Specifies the first object version in a range to be collapsed.

`-from`

Specifies the last object version in a range. The collapse is performed from the last object in a range to the first, inclusive. (See the first example below.)

Examples

In the `ico_616-1` project, `bufcolor.c` has the following version history: 1 --> 2 --> 3 --> 4 --> 5, where version 1 is in the *integrate* state, versions 2, 3, and 4 are in the *checkpoint* state, and version 5 is in the *working* state.

- Collapse `bufcolor.c-4`.

```
ccm collapse bufcolor.c-4 -from bufcolor.c-5
Starting Collapse Process...
Unable to remove bufcolor.c-5:csrc:1 : it is a member of a project.
bufcolor.c-4:csrc:1 removed.
Collapse complete with 1 success and 1 failure.
```

Note that version 5 was not removed. You cannot remove it because it is a member of a project. The new version history of `bufcolor.c` will be 1 --> 2--> 3 --> 5 and version 5 will remain a working version.

- Collapse `bufcolor.c-3` only.

```
ccm collapse bufcolor.c-3
bufcolor.c-3:csrc:1 removed.
```

You would see the following version history for the `bufcolor.c` objects in the `ico_616-1` project: 1 --> 2 --> 5.

Suppose you needed to make changes to `bufcolor.c`, so you checked out, and then checked in a few versions of this object. The version history for the `bufcolor.c` objects in the `ico_616-1` project is now: 1 --> 2 --> 5 --> 6 --> 7 --> 8 (version 1 is in the *integrate* state, versions 2, 5, 6, and 7 are in the *checkpoint* state, and version 8 is in the *working* state).

- Collapse all of the remaining checkpointed versions of the `bufcolor.c` objects in the `ico_616-1` project.

```
ccm collapse bufcolor.c -all
Starting Collapse Process...
Unable to remove bufcolor.c-1:csrc:1 : no write access.
bufcolor.c-2:csrc:1 removed.
bufcolor.c-5:csrc:1 removed.
bufcolor.c-6:csrc:1 removed.
bufcolor.c-7:csrc:1 removed.
Unable to remove bufcolor.c-8:csrc:1 : it is a member of a project.
Collapse complete with 4 successes and 2 failures.
```

  You would see the following version history for the `bufcolor.c` objects in the `ico_616-1` project: 1 --> 8.

- Collapse all products over 10 days old that are not used in any project. You must be in the *ccm_admin* role to collapse non-modifiable objects

  1. Query for the products.

```
ccm query "is_product=TRUE and not is_bound()
and create_time<time('-10:0:0:0')"
```

  2. Collapse the product objects.

```
ccm collapse @
```

Related topics

- checkpoint command

conflicts command

Synopsis

```
ccm conflicts [-r|-recurse] [-t|-tasks] [-v|-verbose] [-noformat] [-nowrap]
project_spec
```

Description and uses

The `conflicts` command displays the conflicts for a project whose update properties are set to update using tasks and a baseline.

If your project's update properties are set to update using object status and you use this command, you will receive a warning message informing you that, in order to use this command, your project must be set up to update using tasks.

For detailed information about conflicts and how they are identified, see Conflict detection.

Options and arguments

project_spec

Specifies the project whose conflicts you want to show.

`-noformat`

Specifies that the command's output is not wrapped, is not formatted, and contains one line for each object version and its conflicts. Each field is separated by a <tab> character, which allows parsing of fields within each line.

This option also enables you to filter the output so that you can, for example, remove lines that contain occurrences of the "No Task" conflicts and keep lines that contain occurrences of "Parallel" conflicts.

`-nowrap`

Specifies that the command's output is not wrapped and contains one line for each object version and its conflicts.

This option also enables you to filter the output so that you can, for example, remove lines that contain occurrences of the "No Task" conflicts and keep lines that contain occurrences of "Parallel" conflicts.

By default, the output from the `conflicts` command wraps at 80 characters.

`-r|-recurse`

Detects conflicts for all projects and subprojects in the project hierarchy topped by the specified project.

`-t|-tasks`

Specifies that task conflicts be displayed. By default, object conflicts are displayed.

`-v|-verbose`

Specifies that you want additional conflict detection messages output.

Example

- Show the conflict detection information for the `toolkit-2` project.

```
ccm conflicts toolkit-2

Project: toolkit-2
Object Ver.  Task Conflicts
-----------  ---- ------------------------------------------------
main.c-0.1.1 57   Implicitly required but not included - parallel
draw.c-4     117  Explicitly specified but not included - parallel
init.c-2             No task
```

copy_project command

Synopsis

```
ccm copy_project|cp|checkout -p|-project|co -p|-project
                [-purpose purpose_spec]
                [-platform platform]
                [-release release|as_is|none]
                [-reconf tasks|os]
                [-subprojects]
                [-versions "old_ver:new_ver,old_ver:new_ver,..."]
                [-t|-to version]
                [-c|-comment "string"]
                [-ce|-commentedit]
                [-cf|-commentfile file_path]
                project_spec
```

Copy and Set Work Area Properties

```
ccm copy_project|cp|checkout|co copy_project_options
                [-cb|-copy_based|-not_copy_based|-ncb  (UNIX only)]
                [-rel|-relative|-nrel|-not_relative]
                [-path|-set|-setpath] absolute_path
                [-mod|modifiable_wa] [-nmod|not_modifiable_wa]
                [-tl|-translate|-ntl|-no_translate]
                [-wa|-maintain_wa|-nwa|-no_wa]
                [-wa_t|-wa_time|-nwat|-no_wa_time]
                [-u|update|-no_u|-no_update]
                project_spec
```

Copy and Update a New Project

```
ccm copy_project|cp|checkout -p|-project|co -p|-project copy_project_options
                [-u|-update|-no_u|-no_update]
                project_spec
```

Description and uses

Use the `copy_project` command to create a modifiable version of a project or project hierarchy. By default, when you copy a project, it is created in the database and a work area is created automatically. You can set work area properties at the time you copy the project. The `copy_project` command functions the same as the `checkout` command with the `-project` option, and the `copy_project` operation was referred to as the `checkout -project` operation in prior releases.

When you specify the `-p` option, the specified project (or the entire project hierarchy) is copied.

When you copy a project from a static (non-modifiable) project and do not copy subprojects, and the subprojects have relative work areas, new copies of those subprojects' work areas are created in the appropriate locations within the work area of the project being copied. This enables developers to reuse static subprojects that have relative work areas.

Static work areas are not maintained and cannot be reconciled with the database; they are ignored during reconcile. Sync'ing a static work area will replace any files that have been modified with files from the database. Copying a project with a static work area will leave the original work area in place; you must reconcile it to discard or keep changes.

Options and arguments

`-c|-comment "`*`string`*`"`

Specifies the comment string.

`-cb|-copy_based (UNIX only)`

Makes the work area copy based.

See work_area command for more information.

`-ce|-commentedit`

Brings up your default editor to enter the comment.

`-cf|-commentfile` *`file_path`*

Uses the comments from the specified file. If you specify both a comment string and comment file, the comments are merged, with the comment string following the comments from the file.

`-mod|-modifiable_wa`

Specifies that the work area can be modified.

`-nmod|-not_modifiable_wa`

Specifies that the work area is not modifiable.

`-no_u|-no_update`

Specifies that the project is not updated when it is copied.

`-not_copy_based|-ncb (UNIX only)`

Makes the work area link-based.

See work_area command for more information.

`-ntl|-no_translation`

Indicates that ASCII files should not be translated when they are copied between Windows and UNIX within the project's work area.

See work_area command for more information.

`-nrel|-not_relative`

On Windows, makes subprojects' work areas absolute instead of relative to the parent project's work area. This is the default when you first create a project.

On UNIX, causes links to be used for subprojects and makes subprojects' work areas absolute instead of relative to the parent project's work area. This is the default when you first create a project.

See work_area command for more information.

`-nwa|-no_wa`

Causes the work area not to be maintained (that is, disconnects your work area from the database).

See work_area command for more information.

`-nwat|-no_wa_time`

Sets the time stamps of the files in the project's work area to reflect the time the files were copied into the work area, rather than the last modification time stored in Rational Synergy.

See work_area command for more information.

`-platform` *`platform`*

Enables you to specify the platform value that is set on the project or project hierarchy that you are copying. The platform choices are listed in the *`CCM_HOME`*`\etc\om_hosts.cfg` file (Windows) or `$CCM_HOME/etc/om_hosts.cfg` file (UNIX) in your Rational Synergy installation area.

By default, the current platform value for each project in the hierarchy is copied to its new version. You can use this option only with the `-p` option.

`-p|-project` *`project_spec`*

Copies a project, or if specified with the `-subprojects` option, the entire project hierarchy is copied.

`-purpose` *`purpose_spec`*

Specifies the purpose to associate with the specified project.

The purpose choices are listed in the Project Purpose Table in synergy classic and include all of the purpose values that are being used in your database.

If you are working as a developer when you copy the project, this option defaults to `Insulated Development`, but you can specify `Shared`, instead. If you are working as a build manager or in the *ccm_admin* role, `Integration Testing` is the default setting.

`-reconf tasks|os`

Enables you to specify whether the project or project hierarchy you are copying is updated using a baseline and tasks (`tasks`) or the state of the candidate objects (`os`). (The latter is the only method used in releases previous to Synergy 4.2.)

`-rel|-relative`

Makes the work area path relative to the parent project's path. You can set this option only if the project is used in one place. After you set it, this project cannot be used as a subproject in any other project.

`-release` *`release`*

Enables you to specify the release value that is set on the project or project hierarchy that you are copying. The release choices include all of the release values that are being used in your database, `as is`, or `none`.

By default, the current release value for each project in the hierarchy is copied to its new version.

`-subprojects`

Causes all subprojects in the specified project's hierarchy to be copied.

`-t|-to` *`file_spec|version`*

Enables you to specify the version of a new project or project hierarchy.

If you are copying a project, you can specify the version only. If you are copying a hierarchy of projects, the new version is used for the project as well as its subprojects. Use the `-versions` option to map new versions to old versions of projects in the hierarchy. The `-to` and `-versions` options are mutually exclusive. Also, if you do not specify the `-to` or `-version` option, the default next version is computed automatically using a Rational Synergy built-in algorithm.

`-tl|-translate`

Indicates that ASCII files should be translated when they are copied between Windows and UNIX within the project's work area.

See work_area command for more information.

`-u|-update`

Specifies to update the project when it is copied. The project is copied without a work area and is then updated, respecting the project grouping's setting that indicates whether the baseline and tasks should be refreshed. Then work area maintenance is enabled, meaning that a work area is maintained for the project.

`-versions "`*`old_ver`*`:`*`new_ver`*`,`*`old_ver`*`:`*`new_ver`*`,..."`

The default next version is computed using a Rational Synergy built-in algorithm. (In most cases the current version is incremented by "1.") To change the next version, map the old version to the new version using the syntax shown.

If you are copying a project hierarchy, each mapping applies to all projects in the hierarchy that currently have that value. If *`new_ver`* is `NoCopy`, projects with the corresponding *`old_ver`* are not copied.

Use the `-to` option to specify the same version for all new projects that you are copying. The `-to` and `-version` options are mutually exclusive.

`-wa|-maintain_wa`

Causes the work area to be maintained (that is, synchronizes the work area and keeps it synchronized).

See work_area command for more information.

`-wat|-wa_time`

Sets the timestamps of the files in the project's work area to show the Rational Synergy modification time rather than the time the files were copied into the work area.

See work_area command for more information.

Examples

- Copy a new version of the `projA-3` project.

  ```
  ccm copy_project -c "test projA" projA-3
  ```

- Copy a new *working* project hierarchy from an existing project hierarchy. Set the versions of all of the projects to your name.

  ```
  ccm copy_project toolkit-int -subprojects -to bill
  ```

- Copy a new *prep* project hierarchy for system testing. Set the release and platform values and versions.

  ```
  ccm copy_project tool_top-1.0 -subprojects -release 2.0 -platform
  win32 -purpose sqa -versions
  "1.0:sqa,win16_1.0:win16_sqa,win32_1.0:win32_sqa"
  ```

- Modify a top-level project's version and propagate the change to its subproject versions.

  ```
  ccm copy_project top_project_spec -subprojects -to version
  ```

Related topics

- checkin command

copy_to_file_system command

Synopsis

`ccm cfs|copy_to_file_system|wa_snapshot` *`project_spec`* `[`*`project_spec`*`] [-p|-path` *`path`*`] [-r|-recurse]`

Description and uses

The `copy_to_file_system` command enables you to make a copy of a non-writable project in your work area.

You cannot maintain and reconcile the project after it is created.

A project copied to your work area has the following characteristics:

- always copy-based, never link-based
- files are read-only
- file modification time is set to the time the copy is created
- can be created on a project that does not have a work area

Options and arguments

`-p|-path` *`path`*

Specifies the path to which the copied project is written. The path defaults to the default work area path; (`ccm_wa\database_name` on Windows, or `ccm_wa/database_name` on UNIX in your home directory).

> **Note** if a path is not specified, the path will default to the default work area path template. Also, the path must be empty and the directory must not contain files.

`project_spec [project_spec]`

Specifies the project to be copied.

`-r|-recurse`

Creates copied projects for the subprojects as well as the selected project (`ccm_wa\database_name` on Windows, or `ccm_wa/database_name` on UNIX in your home directory).

> **Note** This option will create work area copies for the specified project(s) and all subprojects. If this option is not on, subprojects are ignored.

Example

Create a copied project in your work area for project list `proj1-1 proj2-2`:

```
ccm copy_to_file_system -path C:\ccm_wa\ccm_docs proj1-1 proj2-1
```

create command

Synopsis

```
ccm create [-t|-type type] [-task task_number]
           [-c|-comment "string"] [-ce|-commentedit]
           [-cf|-commentfile file_path]
           file_spec [file_spec...] [-v version]
ccm create [-t|-type project]
           [-c|-comment "string"] [-ce|-commentedit]
           [-cf|-commentfile file_path]
           [-release release] [-plat|-platform platform]
           [-purp|-purpose purpose_spec]
           [-task task_number]
           [-reconf|-reconfigure tasks|os]
           [-wa|-maintain_wa|-nwa|-no_wa] [-set]
           project_spec [project_spec...]
ccm create -t|-type project
           [-c|-comment "string"] [-ce|-commentedit]
           [-cf|-commentfile file_path]
           [-release release] [-plat|-platform platform]
           [-purp|-purpose purpose_spec]
           [-task task_number]
           [-reconf|-reconfigure tasks|os]
           -r|-root dir_spec -v version
```

Description and uses

The `create` command creates a new object and adds it to the current Rational Synergy project in the following ways.

- When you create a new file or directory, it is added to the current directory, which must be part of a Rational Synergy project.
- When you create an object in a non-shared project, its default state is *working*. When you create a file or directory in a shared project, its default state is *visible* if it is a non-product, and *shared* if it is a product.
- When you create a new object in a non-writable directory, a new directory version is checked out automatically.

  If you are in a shared project and your current directory is non-modifiable, the directory is checked out and associated automatically with the default (or specified) task and is checked in to the *integrate* state. You can disable the automatic check-in feature by setting `shared_project_directory_checkin` to `FALSE` in your initialization file. (See shared_project_directory_checkin.)
- When you create a new project, it is created as a floating object, but you can make it a subproject in an existing Rational Synergy project by using the `use -p` command. (See Caveats.)

- When you create a project, Rational Synergy creates a work area for it automatically. By default, the work area is located in `My Documents\Synergy\ccm_wa\`*`database`*`\`*`project_name-version`* (Windows) or `ccm_wa/`*`database`*`/`*`project_name-version`* (UNIX) in your home directory. (See work_area command for how to change your work area location.)
- When you use the `-t project -r `*`dir_spec`* `-v `*`version`* options to create a project, Rational Synergy uses the existing *`dir_spec`* as the root directory for the new project, and the project takes the name of the *`dir_spec`* object.
- To add members to a directory, it must be writable (that is, checked out). If you try to create an object in a non-modifiable directory, Rational Synergy checks out the directory automatically. You will need to check in the directory and the new object to make the new object available to other users.

Options and arguments

`-c|-comment "`*`string`*`"`

Specifies the comment string.

`-ce|-commentedit`

Brings up an editor to enter the comments.

`-cf|-commentfile `*`file_path`*

Takes the comments from the specified file. If both a comment string and comment file are specified, the comments are merged, with the comment string following the comments from the file.

`file_spec`

Specifies the name of the file to be created.

`-nwa|-no_wa`

Causes the work area not to be maintained (that is, disconnects your work area from the database).

`-plat|-platform `*`platform`*

Specifies the platform of the project to be created.

`project_spec`

Specifies the name of the project to be created. The new project will be displayed automatically if it is not a subproject.

`-purpose` *purpose_spec*

Specifies the purpose name or member status of the project to be created. Use `ccm project_purpose -show` to view a list of all project purposes.

`-r|-root` *dir_spec*

Specifies that the *dir_spec* will be the root directory for the new project. The new project's name is taken from the root directory's name.

`-reconf|-reconfigure tasks|os`

Specifies how the new project will be updated (reconfigured).

`-release` *release*

Specifies the release of the project to be created.

`-set`

Sets the work area path. This option can be used only with the `-wa` option.

`-t|-type` *type*

Specifies the type of the new object. If you do not specify a type, the default is calculated from the extension (for example, a `.c` object defaults to a `csrc` type).

Use the show command to view a list of valid types.

`-task` *task_number*

Associates the newly created objects with the specified task. Also associates a newly checked-out directory with the task if the object was created in a read-only directory.

If the current (default) task is set and you do not specify a different task, the objects you are creating or checking out are associated with the current task automatically.

`-v` *version*

Specifies the version of the new project. You must use the `type` and `root` options with this option.

When allow_delimiter_in_name is set to `FALSE`, you can include the version in the object name. For example:

```
ccm create foo-1
```

However, if `allow_delimiter_in_name` is set to `TRUE`, you must use the version option. For example:

```
ccm create foo -v 1
```

`-wa|-maintain_wa`

Causes the work area to be maintained (that is, synchronizes the work area and keeps it synchronized).

Examples

- Create an initial project called `proj1` in the work area.

  ```
  ccm create -t project proj1
  ```

- On Windows, create a new C source object called `sort.c` in the `utils\sym_tool` directory.

  ```
  ccm create -type csrc utils\sym_tool\sort.c
  ```

- On UNIX, create a new C source object called `sort.c` in the `utils/sym_tool` directory.

  ```
  ccm create -type csrc utils/sym_tool/sort.c
  ```

- Create a new directory object called `testcase` under the current directory.

  ```
  ccm create -t dir testcase
  ```

- Create a new project from the GUI directory with a version of final.

  ```
  ccm create -t project -v final -r GUI
  ```

- Create an initial project and maintain a work area.

  ```
  ccm create -t project -c "test" -wa -set "/tmp" testwa-1.0
  ```

- Change a Rational Synergy directory into a subproject under the current directory.

  1. Create the project.

     ccm create -t project -purpose *project_purpose* -release *release_value* -r *directory_name* -v *version* [*project_create_options*]

  2. Then, replace the directory with the subproject. First, unuse the directory.

     ccm unuse *directory_name*

  3. Use the new subproject.

```
ccm use -p project-version
```

Caveats

You cannot create a new project by using the `-r dir_spec` option at the top-level project's root directory. Use this option only on subdirectories.

Related topics

- delete command
- show command
- use command

dcm command

Synopsis

Initialize

```
ccm dcm -init -dbid|-database_id database_id
                        [-delim "single-character_delimiter"]
                        [-description description]
                        [-location location]
                        [-admin_info admin_info]
```

Add

```
ccm dcm -add -ts|-transfer_set "transfer_set_name"
                        [-h|-history] | [-nh|-no_history]
                        file_spec [file_spec...]
```

Change Database ID and Update Affected Objects

```
ccm dcm -change -dbid|-database_id database_id -convert
```

Change the DCM Delimiter and Update All Objects

```
ccm dcm -change [-from_delim|-from_delimiter delimiter]
                 -delim|-delimiter delimiter
```

Change Database ID Without Updating Any Objects

```
ccm dcm -change -dbid|-database_id database_id
```

Change Directory Project Instance

```
ccm dcm -change_dir_project_instance |
        -cdpi project_name instance dirobjectspec1 [...dirobjectspecN]
```

Convert the Database ID of Objects Created in Another Database

```
ccm dcm -change -from_dbid|-from_database_id fromdbid
        -to_dbid|-to_database_id todbid
```

Create a Database Definition

```
ccm dcm -create -dbid|-database_id database_id
                        [-desc|-description description_of_database]
                        [-tm|-transfer_mode transfer_mode_name]
                        [-ar|-automatic_receive|-noar|-noautomatic_receive
                        [-rb|-run_in_background|-norb|-norun_in_background]
                        [-host hostname]
                        [-os|-operating_system UNIX|Windows]
                        [-path path]
                        [-ccm_home path]
                        [-zip|-nozip]
                        [-tp|-transfer_path transfer_path]
                        [-ga|-generate_allowed|-noga|-nogenerate_allowed]
                        [-hidden|-nohidden]
                        [-handover_dbid|-handover_database_id dbid]
                        [-location location]
                        [-admin_info admin_info]
```

Create a Transfer Set

```
ccm dcm -create -ts|-transfer_set "transfer_set_name"
                        [-email email_address]
                       [-crsc|-change_request_scope change_request_scope_name]
                        [-crq|-change_request_query query_expression]
                        [-exclude_products|-noexclude_products]
                        [-exclude_imported_objects|
                        -noexclude_imported_objects]
                        [-exclude_types "list_of_types"]
                        [-exclude_typedefs|-noexclude_typedefs]
                        [-local_parallel|nolocal_parallel]
                        [-ferp|-noferp]
                        [-rsc|-release_scope release_scope_name]
                        [-rq|-release_query release_query_string]
                        [-cumrsc|-cumulative_release_scope|-nocumrsc| -
nocumulative_release_scope]
                        [-dir]
                        [-ib|-include_baselines|-noib|-noinclude_baselines]
                        [-ep|-email_policy policy]
                        [-exclude_nct|-exclude_non_completed_tasks|
                        -noexclude_nct|-noexclude_non_completed_tasks]
                        [-exclude_db_info|-noexclude_db_info]
                        [-cumulative|-nocumulative]
```

Delete a Database Definition

```
ccm dcm -delete -dbid|-database_id database_id
```

Delete a Transfer Set

```
ccm dcm -delete -ts|-transfer_set "transfer_set_name"
```

Generate

```
ccm dcm -gen|-generate -dbid|-database_id database_id
                       [-ts|-transfer_set "transfer_set_name"|-notransfer]
                       [-lg|-last_generated last_generated_value]
                       [-wait|-nowait]
                       [-email email_address|-noemail]
```

Generate and Transfer

```
ccm dcm -gen|-generate -trn|-transfer
                       -dbid|-database_id database_id
                       -ts|-transfer_set "transfer_set_name"
```

Generate, Transfer, and Receive

```
ccm dcm -gen|-generate -trn|-transfer -rec|-receive
                       -dbid|-database_id database_id
                       -ts|-transfer_set "transfer_set_name"
```

Modify a Database Definition

```
ccm dcm -modify -dbid|-database_id database_id
                       [-desc|-description description_of_database]
                       [-tm|-transfer_mode transfer_mode_name]
                       [-ar|-automatic_receive|-noar|-noautomatic_receive]
                       [-rb|-run_in_background|-norb|-norun_in_background]
                       [-host hostname]
                       [-os|-operating_system UNIX|Windows]
                       [-path path]
                       [-ccm_home path]
                       [-zip|-nozip]
                       [-tp|-transfer_path transfer_path]
                       [-ga|-generate_allowed|-noga|-nogenerate_allowed]
                       [-hidden|-nohidden]
                       [-handover_dbid|-handover_database_id dbid]
                       [-location location]
                       [-admin_info admin_info]
```

Modify a Transfer Set

```
ccm dcm -modify -ts|-transfer_set "transfer_set_name"
                        [-email email_address|-noemail]
                       [-crsc|-change_request_scope change_request_scope_name]
                        [-crq|-change_request_query query_expression]
                        [-exclude_products|-noexclude_products]
                       [-exclude_imported_objects|-noexclude_imported_objects]
                        [-exclude_types "list_of_types"]
                        [-exclude_typedefs|-noexclude_typedefs]
                        [-local_parallel|nolocal_parallel]
                        [-ferp|-noferp]
                        [-dir]
                        [-rsc|-release_scope release_scope_name]
                        [-rq|-release_query release_query_string]
                        [-cumrsc|-cumulative_release_scope|-nocumrsc| -
nocumulative_release_scope]
                        [-ib|-include_baselines|-noib|-noinclude_baselines]
                        [-ep|-email_policy policy]
                        [-exclude_nct|-exclude_non_completed_tasks|-
noexclude_nct|-noexclude_non_completed_tasks]
                        [-exclude_db_info|-noexclude_db_info]
                        [-cumulative|-nocumulative]
```

Modify DCM Settings

```
ccm dcm -m|-modify -settings
                        [-desc|-description description_of_database]
                        [-location location]
                        [-admin_info admin_info]
                        [-default_add_history|-nodefault_add_history]
                        [-default_include_baselines|-
nodefault_include_baselines]
                        [-ignore_maintain_wa|-noignore_maintain_wa]
                        [-update_db_info|-noupdate_db_info]
                        [-keep_typedefs|-nokeep_typedefs]
                        [-event_log_size log_size]
                        [-parallel_checking parallel_check_keyword]
                        [-update_releases release_action_keyword]
                        [-add_receive_control_transition transition]
                        [-remove_receive_control_transition transition]
                        [-no_of_generate_times generate_times]
                        [-no_of_old_generate_times old_generate_times]
                       [-old_generate_time_resolution old_generate_resolution]
                        [-update_rft|-noupdate_rft]
```

Receive

```
ccm dcm -rec|-receive [-a|-all]
                      [-dbid|-database_id database_id]
                      [-dir directory]
                      [-ts|-transfer_set "transfer_set_name"]
                      [-im|-ignore_missing]
                      [-wait|-nowait]
                      [-ic|-ignore_checks|-noic|-noignore_checks]
```

Recompute the Indirect Change Request Members of a Transfer Set

```
ccm dcm -recompute -crs|-change_requests|-problems -ts|-transfer_set
"transfer_set_name" [-dbid|-database_id database_id]
```

Recompute the Members of a Transfer Set

```
ccm dcm -recompute -ts|-transfer_set "transfer_set_name"
```

Reinitialize

```
ccm dcm -init [-dbid|-database_id database_id]
                       [-delim "single-character_delimiter"]
                       [-description description]
                       [-location location]
                       [-admin_info admin_info]
```

Remove an Object from a Transfer Set

```
ccm dcm -remove -ts "transfer_set_name" filespec [filespec...]
```

Show All Database IDs and Descriptions

```
ccm dcm -show -dbid|-database_id -all
```

Show Current DCM Database ID

```
ccm dcm -show -dbid|-database_id
```

Show Database Definition

```
ccm dcm -show -dbid|-database_id database_id
```

Show DCM Properties

```
ccm dcm -show -prop|-properties objectspec1 [ objectspec2 ... objectspecN]
```

Show DCM Settings

```
ccm dcm -show -settings
```

Show Last Generate Time(s)

```
ccm dcm -show -ts|-transfer_set "transfer_set_name"
                      -dbid|-database_id "database_id"
```

Show One Specified Event

```
ccm dcm -show -event_log|-el -index number
                      [-f|-format format]
                      [-info]
                      [-messages]
```

Show Receive Lock

```
ccm dcm -show -receive_lock
```

Show Summary of Events

```
ccm dcm -show -event_log|-el
                      [-f|-format format]
                      [-dbid|-database_id dbid]
                      [-ts|-transfer_set transfer_set_name]
```

Show Transfer Set

```
ccm dcm -show -ts|-transfer_set "transfer_set_name"
                      [-members direct|all][-u]
```

> **Note** The `ccm dcm -show -ts` command displays change request scope and change request query information only if a Distributed Change (DCS) license is available.

Transfer

```
ccm dcm -trn|-transfer [-a|-all] |
                       [-dbid|-database_id database_id]
                       [-ts|-transfer_set "transfer_set_name"]
```

Prerequisites

The current database must be initialized to use distributed change management (DCM), also called Rational Synergy Distributed. For detailed information about this feature, please read the Rational Synergy Distributed manual, which can be downloaded from the Synergy Support Web site.

Description and uses

The `dcm` command generates a transfer package, sends a transfer package to a destination database, receives a transfer package, and adds objects to a transfer set. The `dcm` command's options enable you to perform one or more of these operations.

You must be working as the DCM manager to use the `-add, -create, -gen, -modify, -delete, and -remove` options. You must be working in the *ccm_admin* role to use the `-rec, -init, -change, -modify, and -settings` options.

Options and arguments

`-a|-all`

Sends all transfer sets to their destination databases when used with the `-trn` option, or receives all transfer sets into the current database when used with the `-rec` option.

This option can be used with `-dbid` for the `ccm dcm -show` command, but not for the `ccm dcm -transfer` command. The `-all` option cannot be used with the `-ts` option.

`-add`

Adds the specified objects (single file, project, task, or folder) to the specified transfer set. The transfer set must be defined already.

`-add_receive_control_transition` *`transition`*

Adds a valid state transition to the Receive Control Transitions list. This added state transition will be allowed when receiving an object that is controlled in the current database. The value that you specify for *`transition`* must be in the following form:

`from_state``:`*`to_state`*

where:

`from_state` must be a valid state, and

`to_state` must be a valid state for which a transition exists from the specified *`from_state`*.

`-admin_info` *`admin_info`*

Specifies the contact information of the person who is responsible for DCM administration issues. The value of *`admin_info`* is free-form text that might include one or more names, phone numbers, and e-mail addresses.

`-ar|-automatic_receive`

Specifies that the receive is to be automatically initiated during the generate operation.

`-ccm_home` *`path`*

Specifies the absolute path to the Rational Synergy installation area. Enter a UNC path if you are using a Windows server.

`-change`

Specifies that a change is to be made to the current database ID and/or DCM delimiter. This option requires that the database be protected and no other sessions are running on the database.

`-change_dir_project_instance|-cdpi`

Modifies the directory entry for the named project to reference the specified instance in one or more specified subdirectories. After you upgrade from an earlier release or after you receive a package from an earlier release, this option makes allows you to fix any such directory entries. The directory arguments that are used with the `-change_dir_project_instance` option are standard object specifications that can include query references.

`-crq|-change_request_query` *`query_expression`*

Specifies a change request query expression. For any `change_request_scope` other than `none`, you can define a query expression. You need a DCS license to use this option. If you use this option without a DCS license, the command fails, and an error message is displayed.

`-crsc|-change_request_scope` *`change_request_scope_name`*

Specifies the change requests and associated tasks and objects that are eligible for inclusion in the transfer package (see the *Rational Synergy Distributed* book for further details). You need a DCS license to use this option. If you use this option without a DCS license, the command fails, and an error message is displayed.

The *`change_request_scope_name`* must have one of the following values:

- `none`
- `crs`
- `crs only`
- `crs and tasks`
- `crs_and_tasks`
- `crs_tasks_and_objects`
- `crs, tasks and objects`
- `change_requests`
- `change_requests_and_tasks`
- `change_requests_tasks_and_objects`

— problems
— problems_and_tasks
— problems_tasks_and_objects

-crs|-change_requests

When used with -recompute, this prevents change requests that were automatically added to the transfer set from being sent in the following DCM transfer package as well as the current one being generated.

-convert

Specifies that the change operation is to change the database ID of the current database and update affected objects. This option is typically used on a database whose database ID is not the desired value and, therefore, all objects that were created in the current database need to updated so that they refer to the correct database ID.

-cumulative

Specifies that the change request scope is cumulative. The change request scope and query for transfer sets is always evaluated each time a generate or generate preview operation is performed. However, if cumulative is specified, older members found by previous queries will never be removed. That is, the indirect query-based membership will only be added to and will thus be cumulative.

-cumrsc|-cumulative_release_scope

Specifies that the release scope is cumulative. The release scope and query for transfer sets are always evaluated and older members found by previous queries will never be removed.

-dbid|-database_id *customer_dbid*

Specifies the database id of customers who can receive the package that is to be generated.

-dbid|-database_id *database_id*

Specifies the destination database for the transfer set when used with the -gen or -trn options, or the source destination database of the transfer set when used with the -rec option.

Specifies the database ID of the database that you want to DCM initialize when used with the -init option.

Specifies the new database ID that you want to assign to the current database when used with the `-change` option. To learn about restricted characters, and why they are restricted, see Naming restrictions. If many objects need to be updated, it takes a long time to carry out the `ccm dcm -change` command.

Specifies that only entries for the specified database will be listed when used with the `-event_log` option.

`-default_add_history`

Specifies that objects that are added to a transfer set are added along with their predecessors.

`-default_include_baselines`

Specifies that baselines that are associated with transfer set members are included in transfers sets.

`-delim` *`delimiter`*

Specifies the new delimiter that you want to assign to the current database.

You cannot use characters a-z, A-Z, or 0-9 for the DCM delimiter. You can use "!", "~", or "=" as an alternative delimiter. The default DCM Delimiter is "#", which should be used whenever possible.

If many objects need to be updated, it takes a long time to carry out the `ccm dcm -change -delim` *`delimiter`* command.

The `-delim` option is not mandatory. If omitted, a default value of "`#`" is used.

If you use the `-init` option to reinitialize a DCM database, the `-delim` option, if specified, must match that of the current database.

`-desc|-description` *`description_of_database`*

Specifies a description of the database.

`-dir|-directory`

Sets the generate directory for the transfer set when used with the ccm dcm `-create -ts` or `ccm dcm -modify -ts` commands. The default on transfer set creation is a blank string. A blank string means that the current database's default `dcm -generate` path is used.

`-dir` *`directory`*

Specifies that the DCM package is to be received from the specified directory, instead of the default directory: *`database`*`/dcm/receive`.

`-email` *`email_address`*

Sets the e-mail address of the person or persons who will receive an e-mail notification following a generate or receive. You can define multiple e-mail recipients for the transfer set by separating the addresses with a space or comma. If you want to define e-mail lists, you can set up e-mail aliases or distribution lists by using the facilities of your mail server. To learn how to do this, consult your mail server and operating system

`-ep|-email_policy` *`policy`*

Specifies the e-mail policy that is used during generate operations and transfer operations. The value that you type for *`policy`* is a case-insensitive string that must be one of the following: `Transfer`, `Generate`, or `Always`.

- `Transfer -` Specifies that an e-mail message is sent only when you transfer a non-empty package to the destination database. Moreover, no message is sent if there are no objects to be included when the DCM generate operation is performed.
- `Generate -` Specifies that an e-mail message is sent when you generate and/or transfer a non-empty transfer package. This is the default when a new transfer set is created. However, no message is sent if there are no objects to be included when the DCM generate operation is performed.
- `Always -` Specifies that an e-mail message is sent whenever you generate and/or transfer a non-empty transfer package. This includes occasions when there are no objects to be included when you perform the DCM generate operation or when you generate a package that is not automatically delivered to the destination database.

`-event_log_size` *`log_size`*

Specifies the maximum number of entries in the event log. The *`log_size`* is a positive non-zero integer. When the event log list reaches the value that you specify for *`log_size`*, each new entry replaces the oldest entry.

`-exclude_db_info`

Specifies that information about the current database and known DCM database definitions are excluded from the DCM information file. This option cannot be used with the `-noexclude_db_info` option.

`-exclude_imported_objects`

Specifies that imported objects are to be excluded from the DCM transfer set.

`-exclude_nct|-exclude_non_completed_tasks`

Specifies that tasks that are not completed are excluded from the transfer list. The `-exclude_non_completed_tasks` and `-noexclude_non_completed_tasks` options cannot be used together.

`-exclude_products`

Specifies that product objects are to be excluded from the DCM transfer set.

`-exclude_typedefs`

Prevents the transfer package from containing user-defined type definitions. This option can only be used with the `ccm dcm -create -ts` command, or the `ccm dcm -modify -ts` command. The `-exclude_typedefs` and `-noexclude_typedefs` options cannot be used together.

`-exclude_types` *`"list_of_types"`*

Specifies that the types in the list are to be excluded from the DCM transfer set.

`-ferp`

Specifies that a project's update (reconfigure) properties are fully expanded to include:

- All task objects—even if these objects are members of the project hierarchy
- All folders and tasks in the projects' update properties

The `-ferp` and `-noferp` options cannot be used together. The default on transfer set creation is `-noferp`.

`-file_spec`

Specifies the name of the file, project, folder, or task that you want to add to the transfer set.

`-format` *`format`*

Specifies the command's output format. The as-shipped default format for the event log is as follows:

```
%index %event_time %event_type %status %dbid %ts
```

where:

`%index` is the index number, which is a unique number that identifies the event entry.

`%event_time` is the date and time at which the log file entry was made.

`%event_type` is the type of event that is logged. Possible values for `%event_type` include `Generate`, `Transfer`, and `Receive`.

`%status` is the status of the operation. Possible values for `%status` include `Started`, `Successful`, `Failed`, and `Cancelled`.

`%dbid` is, for generate or transfer events, the destination database ID. For receive events, `%dbid` is the source database ID.

`%ts` is the name of the transfer set.

You can also specify the `%user` keyword to define a field in the event list; where `%user` is the user who performed the operation.

`-from_dbid|-from_database_id` *`fromdbid`*

Specifies that objects referencing the *`fromdbid`* database are to be converted so that they use the database ID of a *`todbid`* database. This does not change the current DCM database ID. If many objects need to be updated, it takes a long time to carry out a command that uses this option.

`-from_delim|-from_delimiter` *`delimiter`*

Specifies the old DCM delimiter that is to be converted. If the `-from_delim` option is omitted, the old DCM delimiter is converted to the new one that you specify.

`-ga|-generate_allowed`

Specifies that the database whose definition you are creating or modifying can be used as the destination database for a DCM generate operation. This is the default when you create a new database definition. This option cannot be used with the `-nogenerate_allowed` option.

`-gen|-generate`

Generates a transfer package for the specified transfer set and destination database pair.

Enter `dcm -gen` to see a usage message.

`-handover_dbid|-handover_database_id` *`dbid`*

Specifies the database through which control is handed over when control is handed over to the specified database. The default value when creating a DCM database definition is a blank string. This means that when you hand control over to a spoke via a hub database, you must specify the hub dbid for the *`dbid`* option value. The

specified *dbid* must be either a known DCM database definition for which a generate is permitted, or a blank string. A blank string means that no handover of control to that database is permitted.

`-hidden`

Specifies that the database ID (dbid) of the database whose definition you are creating or modifying will not appear in dialog list boxes that pertain to databases. This option cannot be used with the `-nohidden` option.

`-h|-history`

Causes the history of each specified object to be included in the transfer set. You can include the histories only for single files or projects, not for folders or tasks. You use this option only if you are using the `-add` option.

`-ib|-include_baselines`

Specifies that any baselines that are associated with objects that are members of the transfer set will be included. This option cannot be used with the `-noib|-noinclude_baselines` option. The default setting when creating a transfer set is determined by the DCM setting of `Default Include Baselines`.

`-ignore_checks`

Allows a DCM receive operation to continue even if checks on the transfer package indicate a potential problem.

> **Caution** Using this option without understanding its implications may cause incorrect results (see the *Rational Synergy Distributed* book for further details).

This option can only be used with the `ccm dcm -receive` command. The `-ignore_checks` and `-noignore_checks` options cannot be used together.

`-ignore_maintain_wa`

Specifies that projects that were created by import or XML import will never have a maintained work area.

`-im|-ignore_missing`

Tells DCM to ignore any missing transfer packages.

> **Caution** This option may result in empty directory entries or missing relationships.

`-index` *`number`*

Specifies the index number of an event. The index number is a unique number that identifies the event entry. The first event that is logged is assigned an index number of 1, the second event that is logged is assigned an index number of 2, and so on.

A positive non-zero index number is the absolute index of a specific event that is in the DCM event log. A zero or negative index number signifies a relative index; where 0 means the last entry, and a negative number is the number of entries back from the last entry.

`-info`

Specifies that you want to display information that is stored in the summary file for the specified event.

`-init`

Initializes the specified database to the specified database ID and DCM delimiter.

You must use the `-dbid` option if you initializing a database that has not previously been DCM initialized. If the database has already been DCM initialized, the `-dbid` can be omitted, or if specified, the value must match that of the current database ID. Similarly, when you use the to `-init` option to reinitialize a DCM database, the `-delim` option, if specified, must match that of the current database.

`-keep_typedefs`

Specifies that type definitions are kept after a receive operation is completed.

`-lg|-last_generated`

Specifies the last_generated time.

The *`last_generated`* argument must have one of the following values:

— `never`
— `current`
— `integer index` where `1` refers to the most recent generated transfer package.

This option is for advanced users only. If you select a timestamp that is not the most recent timestamp, the generated transfer package includes all objects that have changed since that date. Also, the more recent timestamps are removed from the list.

Specifying `current` sets the time stamp as if the transfer package has just been generated, even though it has not. The result is that all members appear to be up-to-

date, and no objects are included in the transfer package. Use `current` when you are setting up a new transfer package to send updates to hub or publisher databases.

> **Caution** The `never` choice causes all previous time stamps to be removed from the list.

When the value of `last_generated` is `never`, the transfer package is generated as if for the first time; the transfer package will include all objects that were included in previous transfers, plus all objects that have changed since the last transfer.

`-local_parallel`

Results in parallel notifications being e-mailed to local owners of parallels of received objects. This is the default when creating a new transfer set. This option can only be used with the `ccm dcm -create -ts` command, or the `ccm dcm -modify -ts` command. The `-local_parallel` and `-nolocal_parallel` options cannot be used together.

`-location` *location*

Specifies the geographic location of the database (for example, `Irvine, California`). The value of *location* indicates which site owns the database.

`-log`

Specifies that captured messages from the log file are displayed in the event log for the specified event.

`-members direct|all`

If `-members direct` is specified, only direct members are displayed when the command is carried out. If `all` is specified, both direct and indirect members are displayed when the command is carried out.

`-messages`

Specifies that you want to display captured messages from the log file for the specified event.

`-mpi|-map_project_instances`

Specifies that project instances in the database whose definition you are creating or modifying are mapped to a value of "1" during a DCM generate operation. This option cannot be used with the `-nomap_project_instances` option. By default, project

instances are not mapped to a value of "1" when you create a new DCM database definition.

`-nh|-no_history`

Prevents the history of each specified object from being included automatically in the transfer set.

To ensure that only the specified version of the object is included in the transfer set, be sure to use this option. Use this option only if you are using the `-add` option.

`-no_of_generate_times` *`generate_times`*

Specifies the number of generate times that DCM stores. The value that you specify for *`generate_times`* must be a positive non-zero integer that is greater than or equal to the number of old generate times.

`-no_of_old_generate_times` *`old_generate_times`*

Specifies the number of old generate times that DCM stores. The value that you specify for *`old_generate_times`* must be a positive non-zero integer that is less than or equal to the number of generate times.

`-noar|-noautomatic_receive`

Specifies that the receive is not to be automatically initiated during the generate operation.

`-nocumulative`

Specifies that the change request scope will not be cumulative. This is the default when a new transfer set is created. The change request scope and query for transfer sets is evaluated each time a generate or generate preview operation is performed. However, if `nocumulative` is specified, objects that were formerly indirect query members of the transfer set that are no longer found by the query are removed from the transfer set. Thus, the change request scope is not cumulative.

`-nocumrsc|-nocumulative_release_scope`

Specifies that the release scope is not cumulative. The release scope and query for transfer sets are always evaluated and older members found by previous queries will be removed.

`-nocumulative_release_scope`

Specifies that the release scope is not cumulative. The release scope and query for transfer sets is always evaluated and older members found by previous queries will be removed.

`-nodefault_add_history`

Specifies that objects that are added to a transfer set are not added along with their predecessors.

`-nodefault_include_baselines`

Specifies that baselines that are associated with transfer set members are not included in transfers sets.

`-noemail`

Specifies that no e-mail parameter is defined on the transfer package.

`-noexclude_db_info`

Specifies that information about the current database and known DCM database definitions are included in the DCM information file. The default on transfer set creation is `-noexclude_db_info`. This option cannot be used with the `-exclude_db_info` option.

`-noexclude_imported_objects`

Specifies that imported objects are to be included in the DCM transfer set.

`-noexclude_nct|-noexclude_non_completed_tasks`

Specifies that tasks that are not completed are included in the transfer list. This option cannot be used with the `-exclude_non_completed_tasks` option.

`-noexclude_products`

Specifies that product objects are to be included in the DCM transfer set.

`-noexclude_typedefs`

Results in the transfer package containing user-defined type definitions. This option can only be used with the `ccm dcm -create -ts` command, or the `ccm dcm -modify -ts` command. This option cannot be used with the `-exclude_typedefs` option.

`-noferp`

Specifies that a project's update properties are not fully expanded to include. that is, the project's update properties do not include:

- All task objects
- All folders and tasks in the projects' update properties

The `-ferp` and `-noferp` options cannot be used together. The default on transfer set creation is `-noferp`.

`-noga|-nogenerate_allowed`

Specifies that the database whose definition you are creating or modifying cannot be used as the destination database for a DCM generate operation. This option specifies that the database whose definition you are creating or modifying will not appear in DCM Generate dialogs and cannot be used in the `ccm dcm -generate` command. This option cannot be used with the `-generate_allowed` option.

`-nohidden`

Specifies that the database ID (dbid) of the database whose definition you are creating or modifying will appear in dialog list boxes that pertain to databases. This option cannot be used with the `-hidden` option.

`-noib|-noinclude_baselines`

Specifies that any baselines that are associated with objects that are members of the transfer set will not be included. This option cannot be used with the `-ib|-include_baselines` option. The default setting when creating a transfer set is determined by the DCM setting of `Default Include Baselines`.

`-noignore_checks`

Prevents a DCM receive operation from continuing if checks on the transfer package indicate a potential problem. For safety reasons, this is the default. This option can only be used with the `-receive` option. The `-ignore_checks` and `-noignore_checks` options cannot be used together.

`-noignore_maintain_wa`

Specifies that projects that were created by import or XML import will have a maintained work area.

`-nokeep_typedefs`

Specifies that type definitions are not kept after a receive operation is completed.

`-nolocal_parallel`

Prevents parallel notifications from being e-mailed to local owners of parallels of received objects. This option can only be used with the `ccm dcm -create -ts` command, or the `ccm dcm -modify -ts` command. The `-local_parallel` and `-nolocal_parallel` options cannot be used together.

`-norb|-norun_in_background`

Specifies that the automatic receive is not to be executed in the background. Therefore, you have to wait for the transfer package to be received before you can continue using your session.

`-noupdate_db_info`

Specifies that DCM database information is not updated during a receive operation.

`-noupdate_rtf`

Specifies that process rules are not updated during a receive operation.

`-nowait`

Causes a transfer package to perform a DCM receive or auto-receive even though another session might be performing a receive. The default is `-wait`.

`-nozip`

Specifies that the transfer package is left as a collection of data files, each of which is to be transferred. The `-zip` and `-nozip` options cannot be used together.

`-old_generate_time_resolution` *`old_generate_resolution`*

Specifies the interval between the old generate times (expressed in units of days). The value that you specify for *`old_generate_resolution`* must be a positive non-zero floating number.

`-os|-operating_system UNIX|Windows`

Specifies the destination database's operating system.

`-parallel_checking` *`parallel_check_keyword`*

Specifies the type of parallel checking that is performed during a DCM receive operation. The values that you can specify for *`parallel_check_keyword`* include `none`, `created`, and `updated`. These values are case sensitive.

`-path` *`path`*

Specifies the absolute path to the destination database, including the database name. Enter a UNC path if you are using an NT server.

`-prop|-properties`

Displays the transfer sets of which the specified object or objects are members, the type of members, and the database to which the object has been sent. This option displays the same information as the Show DCM Properties dialog.

`-rb|-run_in_background`

Causes the automatic receive to be executed in the background so that you do not have to wait for the transfer package to be received before you can continue using your session.

`-rec|-receive`

Causes a transfer package to be received automatically at the destination database when used with the `-gen` or `-trn` options, or starts the receive operation in the current database when used with the `-rec` option.

Receive all transfer packages with the name "*`transfer_set_name`*" (from all databases) by including the `-ts` option and omitting the `-dbid` option. Receive all transfer packages from the specified database by including the `-dbid` option and omitting the `-ts` option.

Enter `dcm -rec` to see a usage message.

`-receive_lock`

If there is a receive lock, displays the details of the receive lock. If there is no receive lock, this option displays no output.

If a DCM receive was started by using the `-nowait` option, the receive may have skipped getting a receive lock. If so, the `-receive_lock` option will not be able to determine whether such a DCM receive is currently running on the current database.

`-recompute`

Updates the transfer set so that it includes any changes to indirect members. When used with `-change_requests`, this prevents change requests that were automatically added to the transfer set from being sent in the following DCM transfer package as well as the current one being generated. To achieve this, perform the recompute of the change request members, wait the DCM sync tolerance time plus 1 second (the default is 61 seconds), then perform the DCM generate.

`-remove`

Removes the specified objects (single file, project, task, or folder) from the specified transfer set. The transfer set must be defined already.

`-remove_receive_control_transition` *`transition`*

Removes the specified *`transition`* from the Receive Control Transitions list. The removed state transition will no longer be allowed when receiving an object that is controlled in the current database. The value that you specify for *`transition`* must be in the following form:

`from_state``:`*`to_state`*

where:

`from_state` must be a valid state that is in the Receive Control Transitions list, and

`to_state` must be a valid state for which a transition exists from the specified *`from_state`*.

`-rq|-release_query` *`release_query_string`*

Specifies that the specified new release query string should be used when creating or modifying transfer sets. This option can be used only with the `ccm dcm -create -ts` and `ccm dcm -modify -ts` commands.

`-rsc|-release_scope` *`release_scope_name`*

Specifies that a new release scope value should be used when creating or modifying transfer sets. This option can be used only with the `ccm dcm -create -ts` and `ccm dcm -modify -ts` commands.

The *`release_scope_name`* must have one of the following values:

```
none
releases
releases_templates
```

releases_and_templates
release and templates.

-settings

Indicates that DCM settings are to be changed when used with the ccm dcm -modify command. Specifies that DCM settings are to be displayed when used with the ccm dcm -show command.

-show

Displays the current database id when used with the -dbid or -database_id options. The output of the ccm dcm -show -dbid *dbid* command includes the following:

- The geographic location of the database.
- The contact information for one or more persons who are responsible for DCM administration of the database.
- The dbid of the handover database (an empty string means that, during a DCM receive operation, any objects that are controlled in that database and received in the current database will never be marked as pending handover. Moreover, these objects will always be eligible for update from that database).
- The current export format.
- The zip setting.
- Whether or not project instances are mapped to "1" during a DCM generate operation.
- The transfer path.

If the transfer mode is specified as either direct or ftp, the zip setting is automatically changed as follows: For direct, the zip setting is changed to OFF, and for ftp it is changed to ON.

The -show option displays a list of DCM settings when used with the -settings option. The output of the ccm dcm -show -settings command includes the following:

- Description of the current database.
- Location of the current database.
- Information about the person who is responsible for DCM administration issues for the current database.
- Default for history setting on DCM add operation (may be overridden by .ini file).
- Number of entries in event log.
- Whether maintain_wa is ignored by default.
- How release definitions are updated during a receive operation.
- Whether type definitions are kept after a receive operation.

- How parallel checking is performed during a receive operation.
- Which state transitions, if any, may be received for an object that is controlled in the current database.
- Whether DCM database information is updated during a receive operation.
- Whether process rules are updated during a receive operation.
- The number of generate times that DCM stores.
- The number of old generate times that DCM stores.
- The interval between the old generate times (expressed in units of days).

The `-show` option displays the current transfer set name when used with the `-ts` or `-transfer_set` options. The output of the `ccm dcm -show -ts` command shows the following:

- The setting of the `-ferp|-noferp` option.
- The value of the `-dir` option. If this directory is physically stored as a blank string, the Generate Directory is displayed as the current database's default `dcm/generate` directory.

`-tm|-transfer_mode` *`transfer_mode_name`*

Specifies the mechanism used to send a transfer package to the destination database.

The *transfer_mode* argument must have one of the following values:

— `manual | manual_copy`
— `cp | copy | local_copy`
— `ftp | file_transfer_protocol`
— `rcp | remote_copy`
— `user | user_defined`

The transfer modes are as follows:

Manual Copy - Use a manual method, such as tape, to send the transfer package to the destination database. Note that `manual_copy` is a transfer mode that generates the transfer package without sending it.

Local Copy - Send the transfer package to a database that is accessible using a `copy` command. The database is accessible using a copy command if the sending database's engine can copy to the destination database's `database_dir/dcm/receive` directory.

File Transfer Protocol - Send the transfer package using `ftp`. The `ftp` login and destination directory must be set up in advance of any transfers.

Remote Copy - Send the transfer package to a database that is accessible using an `rcp` command. The database is accessible using an `rcp` command if the sending database's engine can `rcp` to the destination database's `database_dir/dcm/receive` directory.

User Defined - Send the transfer package using your own mechanism.

`-to_dbid|-to_database_id` *`todbid`*

Specifies that objects referencing a *`fromdbid`* database are to be converted so that they use the database ID of the *`todbid`* database. This does not change the current DCM database ID. If many objects need to be updated, it takes a long time to carry out a command that uses this option.

`-tp|-transfer_path` *`transfer_path`*

Sends a single transfer package to its destination database when used with the `-gen` option, or sends one or more transfer packages to their destination databases when used with the `-trn` option.

`-trn|-transfer`

Sets the transfer path to the specified value. When you create a DCM database definition, the default `transfer_path` is a blank string.

Send all transfer packages with the name "*`transfer_set_name`*" by including the `-ts` option and omitting the `-dbid` option. Send all transfer packages destined for the specified database by including the `-dbid` option and omitting the `-ts` option.

Enter `dcm -trn` to see a usage message.

`-ts|-transfer_set "`*`transfer_set_name`*`"`

Specifies the name of the transfer set to generate, send, or receive, or to which transfer set to add one or more objects.

Specifies that only entries for the specified transfer set will be listed when used with the `-event_log` option.

The transfer set name must be enclosed in double quotes if it contains one or more spaces.

`-u`

Suppresses automatic numbering of this command's output ("un-numbered").

`-update_db_info`

Specifies that DCM database information is updated during a receive operation. This information is updated from data that is in the DCM Information file .

`-update_rtf`

Specifies that process rules are updated during a receive operation.

`-update_releases` *`release_action_keyword`*

Specifies how release definitions are updated on a DCM receive. The values that you can specify for *`release_action_keyword`* include `none`, `active`, and `inactive`. These values are case sensitive and are described as follows:

- `none`

  Specifies that release definitions are neither created nor updated.

- `active`

  If the DCM transfer package includes release definitions, any release definitions that currently exist in the receiving database are updated, but only new active release definitions are created. If the DCM transfer package includes release table information only (from Synergy 6.2 or earlier), release definitions for releases are created as active releases.

- `inactive`

  If the DCM transfer package includes release definitions, these are created or updated in the receiving database. If the DCM transfer package includes release table information only (from Synergy 6.2 or earlier), release definitions for releases are created as inactive releases.

`-wait`

Causes a transfer package to wait indefinitely until the receiving database has completed the receive. The user is able to interrupt (CTRL+C) the command to abort the operation. The default is `-wait`.

`-zip`

Specifies that the transfer package will be tarred and zipped. The default value of `-zip` is `TRUE`, expect when the transfer mode id `Direct`. This option does not affect the `direct` transfer mode. The `-zip` and `-nozip` options cannot be used together.

Related topics

- dcm examples

dcm examples

Choose from examples for the following operations.

- Add
- Change
- Create
- Delete
- Generate
- Generate and Transfer
- Generate, Transfer, and Receive
- Modify Settings
- Receive
- Show
- Transfer

Add

- Add the `infotec-23` project to the "InfoServer source" transfer set.

```
ccm dcm -add -ts "InfoServer source" infotec-23:project:1
Adding object 'infotec-23:project:1' to transfer set 'InfoServer
source'.
```

You also can use query output to specify object names.

Change

- Change the database identifier to `ldn1` without updating objects.

```
ccm dcm -change -dbid ldn1
```

- Change the database identifier from `jfil` to `sdg1` updating all objects.

```
ccm dcm -change -from_dbid jfil -to_dbid sdg1
...progress messages...
DCM database conversion is complete with no errors.
4 objects had attributes updated.
No objects had directory entries updated.
Your Rational Synergy session must be restarted.
Rational Synergy engine exiting.
```

- Change the DCM delimiter to a new value and update all objects in the current database so that they use the same delimiter.

```
ccm dcm -change -from_delimiter delimiter -delim delimiter
```

Create

- Create a DCM database definition and add it to your DCM destination database definitions.

```
ccm dcm -create -dbid database_id -desc description -tm
transfer_mode_name
```

- Exclude object types from a transfer set (while you are creating the transfer set).

```
ccm dcm -create -ts "transfer_set_name" -exclude_types "list_of_types"
```

- Include products in a transfer set (while you are creating the transfer set).

```
ccm dcm -create -ts "transfer_set_name" -noexclude_products
```

- Exclude imported objects from a transfer set (while you are creating the transfer set).

```
ccm dcm -create -ts "transfer_set_name" -exclude_imported_objects
```

- Exclude database information from a transfer set (while you are creating the transfer set).

```
ccm dcm -create -ts "transfer_set_name" -exclude_db_info
```

Delete

- Delete one or more transfer sets.

```
ccm dcm -delete -ts "transfer_set_name"
```

Generate

- Generate the transfer package for the `Secure transformer layer` transfer set and the `BST` database, and save it to transfer later.

```
ccm dcm -gen -ts "Secure transformer layer" -dbid BST
Computing transfer package...
Computing transfer package for 'Secure transformer layer' going to
database 'BST'...
115 objects will be included in transfer package for 'Secure
transformer layer' going to database 'BST'...
Generating transfer package...
...
DCM data generated to file
 '\\ccmsrv\ccmdbs\appdevdb\dcm\generate\CA#7#BST#865889312.tar.gz'
Updating database...
DCM Generate completed successfully.
```

Generate and Transfer

- Generate and send the transfer package for the `Visual interface include files` transfer set and the `CA` database.

```
   ccm dcm -gen -ts "Visual interface include files" -dbid CA -trn
   Computing transfer package...
   Computing transfer package for 'Visual interface include files' going
to database 'CA'...
   123 objects will be included in transfer package for 'Visual interface
   include files' going to database 'CA'...
   Generating transfer package...
   ...
   Transferring package...
   Transfer successful, clean up in progress...
   Sending transfer package status email...

   DCM generate-transfer email notification has been sent to:
   dcmadmin@company.comUpdating database...
   DCM Generate completed successfully.
```

Generate, Transfer, and Receive

- Generate, send, and automatically receive the transfer package for the `Visual Interface Project` transfer set and `CH` database.

```
   ccm dcm -gen -ts "Visual Interface Project" -dbid CH -trn -rec

   Computing transfer package...
   Computing transfer package for 'Visual Interface Project' going to
database 'CH'...
   335 objects will be included in transfer package for 'Visual Interface
Project' going to database 'CH'...
   Generating transfer package...
   Creating transfer package information files...
   DCM transfer information saved to file '
   \\ccmsrv\ccmdbs\appdevdb\dcm\generate\CA#7#CH#865893092#dcm_info.txt'
   DCM transfer preview information saved to file
'\\ccmsrv\ccmdbs\appdevdb\dcm\generate\CA#7#CH#865893092#dcm_preview.txt'
   Creating transfer package...
   Creating transfer package for 'Visual Interface Project' going to
database CH...
   Compressing transfer data...
   Compressing transfer data for transfer set 'Visual Interface Project'
and database CH...
   Cleaning up temp files...
   DCM data generated to file
   '\\ccmsrv\ccmdbs\appdevdb\dcm\generate\CA#7#CH#865893092.tar.gz'
   Transferring package...
   Transfer successful, clean up in progress...
   Doing remote receive...
   Starting receive remotely. Progress will not update until the receive
completes.
```

```
   Executing command '\\dbsrv\ccm\bin\util\ccm_receive -h dbsrv -d
\\dbhost\ccmdbs\visystem -dbid CA -ts Visual Interface Project
ccm_home\\dbsrv\ccm'
   Receiving all transfer sets 'Visual Interface Project' from database
'CA'...
   Receiving transfer package 'Visual Interface Project' from database
CA...
   Receiving 1 of 1 transfer packages...
   Decompressing data generated 06-09-97 14:51:32 PDT for transfer set
'Visual Interface Project' and database CA...
   Extracting data...
   Extracting data generated 06-09-97 14:51:32 PDT for transfer set
Visual Interface Project' and database CA...
   Importing data...
   Receive complete for transfer package 'Visual Interface Project' from
database CA.
   DCM Receive completed successfully.
   Rational Synergy engine exiting.

   Mon Jun  9 14:51:53 2004

   Receive DCM data done. Remote receive complete
   Sending transfer package status email...
   DCM generate-transfer email notification has been sent to:
   dcmadmin@company.com
   Updating database...
   DCM Generate completed successfully.
```

Initialize

- Initialize the current database by using the database ID `jfil` and the default DCM delimiter ("`#`").

  Include the following in the initialize command.

  — A description of the database.

  — The geographic location of the site that owns the database.

  — The contact information of the person who is responsible for DCM administration issues.

  By entering the preceding data, when you replicate with other databases, a DCM database definition is created that includes `description`, `location`, and `admin_info` fields that are populated with meaningful data.

```
   ccm dcm -init -dbid jfil -description "Development database for
ProductXYZ" -location "Los Angeles, California" -admin_info "Jane Smith,
(206) 555-9090, JSmith@xyz.com"
```

```
...progress messages...


DCM database conversion is complete with no errors.
4 objects had attributes updated.
No objects had directory entries updated.
Your Rational Synergy session must be restarted.
Rational Synergy engine exiting.
```

Modify Settings

- Change the settings for the event log size and the number of old generate times.

```
ccm dcm -modify -settings -event_log_size log_size -
no_of_old_generate_times old_generate_times
```

Receive

- Receive a transfer package from a source database.

```
ccm dcm -rec -dir /vol/dbserver1/dcm/receive/ -ts "transfer_set_name"
-dbid src_database_ID
```

Show

- Show DCM events for database `sdg1`.

```
ccm dcm -show -event_log -dbid sdg1

66 Mon Jul 22 15:16:53 2002 Receive      Successful E   eproj
65 Mon Jul 22 15:15:22 2002 Receive      Successful E   eproj
64 Wed Jul 17 15:27:13 2002 Receive      Successful K   All projects and
related objects for Release rename/1.0 saved on Wed Jul 17 15:23:45 2002
63 Wed Jul 17 15:23:45 2002 Save Offline Successful Any All projects and
related objects for Release rename/1.0 saved on Wed Jul 17 15:23:45 2002
62 Fri Jul 12 14:53:00 2002 Receive      Successful K3  M12251
61 Fri Jul 12 14:50:55 2002 Generate     Successful K3  M12251
60 Fri Jul 12 12:30:44 2002 Receive      Successful E   task completed_in
test
59 Fri Jul 12 12:29:48 2002 Receive      Successful E   task completed_in
test
58 Thu Jun 27 17:42:49 2002 Generate     Successful foo foo
57 Thu Jun 27 16:09:10 2002 Generate     Failed     foo foo
56 Thu Jun 27 15:25:13 2002 Generate     Cancelled  foo foo
55 Thu Jun 27 15:23:19 2002 Generate     Successful foo foo
54 Thu Jun 27 15:22:11 2002 Generate     Cancelled  foo smallexport
53 Wed Jun 26 13:13:42 2002 Generate     Cancelled  K3  R17951
52 Wed Jun 26 13:12:07 2002 Generate     Cancelled  K3  R17951
51 Wed Jun 26 13:10:53 2002 Generate     Successful K3  R17951
50 Wed Jun 26 13:10:20 2002 Generate     Cancelled  K3  R17951
49 Wed Jun 26 13:09:57 2002 Generate     Cancelled  K3  R17951
```

```
48 Tue Jun 25 14:53:48 2002 Generate     Successful K3  smallexport
47  Wed Jun 19 16:08:20 2002 Generate     Successful K3  jre
46  Wed Jun 19 16:08:05 2002 Generate     Successful K3  jre
45  Wed Jun 19 16:06:57 2002 Generate     Failed     K3  jre
44  Fri Jun 14 15:18:50 2002 Receive      Successful E   skipback
43  Fri Jun 14 15:17:27 2002 Generate     Successful E   skip
42  Fri Jun 14 15:02:32 2002 Generate     Successful E   skip
41  Fri Jun 14 10:18:14 2002 Generate     Successful E   jre
40  Thu Jun 13 18:37:15 2002 Generate     Successful E   jre
39  Thu Jun 13 16:08:20 2002 Receive      Started    E   eproj
38  Thu Jun 13 15:30:37 2002 Generate     Successful E   jre
37  Thu Jun 13 13:31:42 2002 Generate     Successful E   jre
36  Mon Jun 10 23:27:56 2002 Save Offline Successful Any Projects named
ccm on Mon Jun 10 23:27:56 2002
35  Mon Jun 10 23:23:29 2002 Save Offline Successful Any Projects named
ccm on Mon Jun 10 23:23:29 2002
34  Thu Jun 06 15:31:19 2002 Generate     Successful K3  test subproject
rename
33  Thu Jun 06 15:29:04 2002 Generate     Successful K3  test subproject
rename
32  Thu Jun 06 13:55:33 2002 Generate     Successful K3  test subproject
rename
31  Tue Jun 04 16:15:29 2002 Generate     Successful K3  folder only
30  Tue Jun 04 16:05:47 2002 Generate     Started    K3  folder only
29  Thu May 30 19:34:15 2002 Generate     Successful K3  testwa
28  Thu May 30 19:32:59 2002 Generate     Successful K3  testwa
27  Thu May 30 19:30:20 2002 Generate     Successful K3  testwa
```

Transfer

- Send transfer packages for all transfer sets to the `BST` database.

  ```
  ccm dcm -transfer -dbid BST
  Transferring all transfer packages with destination database 'BST'...
  Transferring 'Secure transformer layer' generated on 06-09-97 13:48:32
  PDT to database 'BST'.
  Sending transfer package status email...
  DCM generate-transfer email notification has been sent to:
  dcmadmin@company.com
  Transfer completed successfully.
  ```

- Transfer the saved transfer package for the `RDBMS Server API` transfer set (that was generated for the `CH` database).

  ```
  ccm dcm -transfer -ts "RDBMS Server API"
  Transferring all transfer packages 'RDBMS Server API'...
  Transferring 'RDBMS Server API' generated on 06-09-97 14:42:20 PDT to
  database 'CH'.
  Sending transfer package status email...
  ```

```
DCM generate-transfer email notification has been sent to:
dcmadmin@company.com
Transfer completed successfully.
```

delete command

Synopsis

```
ccm del|delete [-repl|-replace] [-scope delete_scope]
               [-t|-task task_number]
               file_spec [file_spec...]
ccm del|delete [-repl|-replace] -force file_spec [file_spec...]
ccm del|delete [-r|-recurse] [-repl|-replace]
               [-t|-task task_number]
               file_spec [file_spec...]
ccm del|delete [-repl|-replace] [-scope delete_scope]
               -p|-project project_spec [project_spec...]
ccm del|delete [-r|-recurse] [-repl|-replace]
               -p|-project project_spec [project_spec...]
ccm del|delete [-r|-recurse] [-repl|-replace] [-h|-hierarchy]
               [-t|-task task_number] [file_spec...]
ccm del|delete [-r|-recurse] [-repl|-replace] [-h|-hierarchy]
               -p|-project project_spec [project_spec...]
```

Description and uses

The `delete` command enables you to delete a specific version of a file or directory or project from a directory and from the database. Additionally, you can delete a project hierarchy from the command line or from the GUI.

An object version can be deleted if it is not a member of a project or if it is only a member of the current project and has no successors.

> **Note** When you delete an object from a non-writable directory, a new directory version is checked out automatically.
>
> If you are in a shared project and your current directory is non-writable, the directory is checked out and associated automatically with the current (or specified) task and is checked in to the *integrate* state. You can disable the automatic check-in feature by setting `shared_project_directory_checkin` to `FALSE` in your initialization file. (See shared_project_directory_checkin.)
>
> For more information, see Shared projects.
>
> **Caution** The delete operation is permanent.

Options and arguments

file_spec

Specifies the name of the file or directory to be deleted.

-force

The -force option suppresses confirmation messages and forces the delete operation to be carried out.

-h|-hierarchy

Causes the operation to delete the entire project hierarchy. Must be used with the -recurse option.

-p|-project *project_spec*

Deletes the specified project.

-r|-recurse

If the target of the command is a directory or a project, this option causes all objects, including the directory and/or project, to be deleted, which will make objects that cannot be deleted floating objects.

When using this option to hierarchically delete objects, the following apply:

- -recurse *file_spec* or -recurse -hierarchy *file_spec* deletes the specified file in the current project. (Rational Synergy ignores both -recurse and -hierarchy if *file_spec* is not a directory.)
- -recurse *file_spec* (where *file_spec* is a directory) alone deletes all object versions in a directory, except subprojects.
- -recurse -hierarchy *file_spec* (where *file_spec* is a directory) deletes all object versions, including subprojects, that reside below the specified directory.
- -recurse -project *project_spec* alone deletes all object versions in a project, except subprojects.
- -recurse -hierarchy -project *project_spec* deletes all object versions, including subprojects, that reside below the specified project.

When used with a *project_spec*, the -recurse option is the same as defining the scope of the delete with the -scope project_and_non-project_members option. The -recurse -hierarchy option is the same as defining the scope of the delete with the -scope entire_project_hierarchy option.

`-repl|-replace`

Deletes an object and replaces it with its predecessor.

`-scope` *`delete_scope`*

Defines the scope of the delete. The valid *`delete_scope`* values are:

- `project_only`
- `project_and_non-project_members`
- `project_and_subproject_hierarchy`
- `entire_project_hierarchy`

The valid values for directory objects are:

- `directory_only`
- `directory_and_non-project_members`
- `entire_directory_hierarchy`

`-t|-task` *`task_number`*

When you delete an object whose parent directory is read-only, a new version of the directory is checked out automatically.

This option associates the newly checked-out directory with a task if the object was deleted from a read-only directory.

If the current task is set and you do not specify a different task, the newly checked-out directory is associated with the current task automatically.

Example

- Delete the `sort.c` file and replace it with the previous version.

```
ccm delete sort.c
Member sort.c-1 deleted from project ico_proj-1
```

Related topics

- create command
- unuse command
- use command

delimiter command

Synopsis

```
ccm delim|delimiter [value]
```

Description and uses

The delimiter is the character that separates the project or object name and version values. Also, you can use it to separate the project name from the version when creating the initial work area path for a project.

The `delimiter` command enables a user in the *ccm_admin* role to change the value of the delimiter character. The default is the dash character (-). You can set the delimiter to any non-restricted character (see Command and argument syntax). Additionally, be sure to read Naming restrictions. When you set the delimiter, you set it for a Rational Synergy database.

The main reason for changing a delimiter is to avoid using a character that is used in object names in the Rational Synergy database. The user in the *ccm_admin* role should ensure that the default delimiter will not conflict with objects that contain the dash character as a part of their name. If this is the case, change the default delimiter **before** users begin to use the database.

> **Note** If you change the delimiter for a database, do so before migrating software under Rational Synergy control. You can change the delimiter at any time, but if you do so when projects already exist in a database, you will need to change the work area paths of those projects that include the delimiter character.
>
> After you change the delimiter and restart the interface, any new project that you create will use the new delimiter that you set. Work area paths for existing projects are not affected.
>
> **Caution** Changing your delimiter might affect your work areas. See work_area command for more information.

You must be in the *ccm_admin* role to use this command.

Delimiter Restrictions

Some characters are forbidden for use as the delimiter by the `ccm delimiter` command. These characters are listed in Naming restrictions.

You can control whether the delimiter is a restricted character. See allow_delimiter_in_name for information about changing restrictions for non-project object names. The delimiter is still restricted for versions, types, instances, and projects.

Options and arguments

value

Specifies the new character to be used as the delimiter for an entire database.

Examples

- The display name, object name, and work area path use the delimiter to separate the name from the version. For example, your project work area might be in the following location.

  Windows:
  ```
  c:\users\linda\ccm_wa\ccmint22\hello-linda
  ```
  UNIX:
  ```
  ~linda/ccm_wa/ccmint22/hello-linda
  ```

  You could change the delimiter between the file name and version to a comma by using the following `delim` command.

  ```
  ccm delim ","
  ```

  When you restart your interface and create a new project (for example, `goodbye,linda`), your project work area is in the following location.

  Windows:
  ```
  c:\users\linda\ccm_wa\ccmint22\goodbye,linda
  ```
  UNIX:
  ```
  ~linda/ccm_wa/ccmint22/goodbye,linda
  ```

- Suppose you want to reference a `csrc` file's work area version of `poly.c,2` even though you have version 3 (`poly.c,3`) in your work area. If your delimiter is set to a comma, you would specify the file as follows:

  Windows:
  ```
  ccm dir poly.c,2
  integrate mary Aug 01 08:07 csrc 1 poly.c,2
  ```
  UNIX:
  ```
  ccm ls -l poly.c,2
  integrate mary Aug 01 08:07 csrc 1 poly.c,2
  ```

  If you tried to specify the file using the dash delimiter, you would see the following error message:

  Windows:
  ```
  ccm dir poly.c-2
  Referenced object version could not be identified: 'poly.c-2'
  ```
  UNIX:
  ```
  ccm ls -l poly.c-2
  Referenced object version could not be identified: 'poly.c-2'
  ```

Related topics

- work_area command

depend command

Synopsis

```
ccm depend -a|-append dependency_file [-f makefile]

ccm depend -w|-write dependency_file [-f makefile]

ccm depend -d|-delete [-f makefile]

ccm depend -s|-show [-f makefile]
```

Description and uses

The `depend` command enables you to update makefile dependencies using a third-party *`make`* tool.

First, generate makefile format dependencies using a third-party tool. Next, execute the `depend` command to copy the generated dependencies to the controlled makefile's dependency attribute.

> **Note** If you do not use the `-f` option, Rational Synergy searches the current directory for a valid, controlled makefile (upper-, lower-, or mixed-case "`makefile`" or "`makefile.mk`") and performs the operation on that makefile object's dependency attribute.

Any user can execute this command.

Options and arguments

`-a|-append`

Appends the dependencies in *`dependency_file`* to any existing dependencies associated with the makefile.

`-d|-delete`

Deletes the dependencies from the makefile's dependency attribute.

`dependency_file`

Specifies the name of the file that contains the dependencies.

`-f` *`makefile`*

Specifies the name of the makefile to which to copy dependencies.

`-s|-show`

Displays the makefile's dependency attribute. Rational Synergy uses the default view method to display the attribute.

`-w|-write`

Saves the dependencies in *dependency_file* for the makefile. This option overwrites any existing dependencies for the makefile.

Examples

- Generate dependencies using your *make* tool, and then save the dependencies to the dependency attribute on the *makefile_name*.

  Windows:
  ```
  mytool foo.c > depend_file
  ccm depend -w depend_file -f makefile_name
  del depend_file
  ```

  UNIX:
  ```
  mytool foo.c > depend_file
  ccm depend -w depend_file -f makefile_name
  rm depend_file
  ```

- Delete the dependencies associated with the *makefile_name*.

  ```
  ccm depend -d -f makefile_name
  ```

- Append a new set of dependencies from the *depend_new_file* to the *makefile_name*.

  ```
  ccm depend -a depend_new_file -f makefile_name
  ```

diff command

Synopsis

```
ccm diff [-vc|-versioncompare] file_spec1 file_spec2
ccm diff [-vc|-versioncompare]
         -p|-project project_spec1 project_spec2
```

Description and uses

The `diff` command shows the differences between files, directories, or projects.

Use this command to do two types of comparisons: a *source compare* (the default) and a *version compare*.

Compare

Shows the differences between source files, directories, or projects. If you perform the `diff` command on directories, a comparison of the lists of non-versioned members is done. For projects, the lists of versioned member files are compared.

Version Compare

Compares other attributes of the files, directories, and projects that are not considered to be the source; for example, the `create_time`, `modify_time`, `name`, and `version`.

Options and arguments

`file_spec1`

Specifies the name of the first file or directory that you want to compare.

`file_spec2`

Specifies the name of the second file or directory that you want to compare.

`-g`

Brings up the graphical compare tool or the appropriate dialog, depending on the type of objects you are comparing. You must enter the names and versions of the objects whose differences you want to view; otherwise, you will receive an error message.

If you start a session with the `-nogui` option, you will not be able to view differences between objects graphically.

`-p|-project`

Shows the differences between projects.

project_spec1

Specifies the name of the first project that you want compared.

project_spec2

Specifies the name of the second project that you want compared.

-vc|-versioncompare

Perform a version compare instead of a source compare. (A source compare is the default.)

Examples

- Compare the current version of `draw.c` with the original.

  ```
  ccm diff draw.c draw.c-1
  ```

- Compare the `tools` and `my_tools` directories.

  ```
  ccm diff tools my_tools
  ```

- Compare the projects `rel-test3.1` and `rel-joe_3.1`.

  ```
  ccm diff -p rel-test3.1 rel-joe_3.1
  ```

- Do a version compare of `draw.c-4` and `draw.c-5`.

  ```
  ccm diff -vc draw.c-4 draw.c-5
  ```

- Do a version compare of projects `rel-test3.1` and `rel-joe_3.1`.

  ```
  ccm diff -vc -p rel-test3.1 rel-joe_3.1
  ```

Related topics

- merge command

dir command

Synopsis

```
ccm dir [-m] [-w] [-s] [-f|-format "format_string"] [file_spec...]
```

Description and uses

The `dir` command operates only on Windows operating systems.

The `dir` command lists the contents of a directory in a work area. Enter this command without any options to view objects in the long format, which contains the status, owner, last modification time, type, instance, name, version, and task.

By default, the output consists of a list of objects and their associated projections in the file system.

The `dir` command displays two categories of files: objects under Rational Synergy control and files that exist in the file system only. To find out how to display these files, see the `-w` option and the `-m` option.

Options and arguments

`-f|-format "format_string"`

Specifies the format of the output. The format only applies to controlled files. The required string uses keywords and literal text, such as:

```
%displayname %owner
```

A keyword can be built-in (`%fullname`, `%displayname`, `%objectname`) or the name of any existing attribute such as `%modify_time` or `%status`.

See Built-In keywords for a list of keywords.

When `%path` is specified in the format string, all objects are displayed with an absolute work area path. If the work area is not visible, the path is computed.

`file_spec`

Specifies the file(s) to be displayed.

`-m`

Shows files that are controlled and uncontrolled. Uncontrolled files are those that are in the work area, but not in the project.

This option causes the following marks to display before the output, when appropriate:

- LC (local copy)

Denotes files that are in the project and in a copy-based work area. (All Windows client users work in a copy-based work area.)

- NS (not sync'd)

Denotes files that are in the project, but not in the work area. This occurs when files are created in the project, but are not sync'd out to the work area, or if the work area copy is deleted.

If most of the files in your work area are displayed with this mark, perform a reconcile operation. For information on the reconcile feature, see the reconcile command.

- UC (uncontrolled)

Denotes files that are in the work area, but not in the project. Note that to view uncontrolled, marked files, you must use the `-m` option with the `-w` option.

`-s`

Displays subdirectory members recursively. The command does not recurse into subprojects.

`-w`

Lists the file name and version only, in an unformatted column.

Examples

- List the files that are not controlled by Rational Synergy.

```
ccm dir -m
(UC) working  bill Jan 11 18:09        csrc 1 cli.c-2 543
(UC) working  bill Jan 11 18:09        csrc 1 input.c-1 432
(UC) working  bill Jan 11 18:09        csrc 1 simple.c-2 543
(UC) working  bill Jan 11 18:09        csrc 1 main.c-1 432
(UC) drwx        0 Feb 02 11:11        images
     released   bob Jan 11 18:11          dir 1 scripts-2 440
```

- List the current directory in the long format. (Files preceded by LC are local copy files.)

```
ccm dir
(LC)  integrate  paul  Dec 24  16:48  makefile    4  Makefile-1
(LC)  integrate  paul  Dec 24  16:48      csrc    1  callback.c-1
(LC)  integrate  paul  Dec 24  16:46      csrc    1  error.c-1
(LC)  integrate  paul  Dec 24  16:46      csrc    1  gui.c-1
(LC)  integrate  paul  Dec 24  16:46      csrc    1  info.c-1
(LC)  integrate  paul  Dec 24  16:47      csrc    1  init.c-1
(LC)    working  paul  Dec 24  17:17      csrc    2  main.c-3
(LC)  integrate  paul  Dec 24  16:48      csrc    1  output.c-1
(LC) integrate    paul  Dec 24 16:49     csrc   1  release.c-1
```

- In the current directory, list the file name and version for all objects.

```
ccm dir -w
ext_incl-1
incl-1
src-1
```

- In the current directory, show all members, including subdirectories.

```
ccm dir /s

integrate ann Jun 19  2003      dir  J#1 include,2 J#5565
(LC) integrate pat Jan 26 15:41 makefile  J15 Makefile.pc,#7 J#6103
   released    ann Jan 16  2001      dir J#12 src,1 J#120

include:
(LC) integrate pat Jan 26 15:42 makefile J#1 make_include.pc,13 J#6103

src:
(LC) integrate joe Mar 27  2003 java J#1 Main.c,6 J#5339
```

- In the current directory, show the absolute paths for all objects.

```
ccm dir /f "%displayname %type %path"

VersionedObject.java,10 java C:\joe\ccm_wa\ccm_java\objectapi-
joe\objectapi\src\com\Rational\cm\objectapi\VersionedObject.java
```

edit command

Synopsis

```
ccm edit file_spec [file_spec...]
```

Description and uses

The `edit` command enables you to edit the specified source file.

Use this command to view a version of a file that is not currently a member of the directory or to invoke the type-based editor on the object. The default editor is used to edit the file.

> **Note** When you edit a file (Windows), or edit your file in a copy-based work area (UNIX), the corresponding database object is not updated automatically with your changes. Reconcile your work area regularly when your are editing and saving a files.

Options and arguments

file_spec

Specifies the file that you want to edit.

Example

Edit version 8 of the `log.c` file. To edit an object, it must be writable by you.

```
ccm edit log.c-8
```

Caveats

On Windows, modifiable files can be edited only from within a project with a visible work area.

On UNIX, only files that are modifiable by the current user can be edited.

Related topics

- view command
- reconcile command
- typedef command
- cli.text_editor
- cli.text_viewer

expand command

Synopsis

```
ccm expand [-c] [-s suffix] makefile [makefile...]
ccm expand -a|-all [-c] [-s suffix]
```

Description and uses

The `expand` command converts a Rational Synergy Makefile to a stand-alone makefile.

Options and arguments

`-a|-all`

Expands all objects of type `makefile` in the current project.

`-c`

Controls the expanded makefile under Rational Synergy.

`makefile`

Specifies one or more makefiles to expand. You do not need to specify any makefiles if you are using the `-a` option.

`-s` *`suffix`*

Specifies the expanded makefile's suffix. (The default suffix is `.out`.) If the `%platform` string is found in the suffix, `expand` replaces that string with the `platform` attribute from the project or makefile. The makefile `platform` attribute overrides the project `platform` attribute.

Example

- Generate a stand-alone makefile from a makefile named `Makefile`.

  ```
  ccm expand Makefile -s
  ```

The default name for an expanded makefile is *`makefile_name`*`.out`.

export command

Synopsis

```
ccm export [-r|-recurse] [-t|-to pathname] [-q|-quiet]
           [-delimiter delimiter_value]
           file_spec [file_spec...]
ccm export [-r|-recurse] [-t|-to pathname] [-q|-quiet]
           [-delimiter delimiter_value]
           [-leave_wa_paths|-lwp]
           -p|-project project_spec [project_spec...]
```

Prerequisite

User *ccm_root* must be able to write to the export directory because the Rational Synergy engine process performs the export, and that process runs as user *ccm_root*.

Description and uses

The `export` command exports objects from the Rational Synergy database to the file system in the Rational Synergy import/export format.

Any user can execute this command.

Note The export directory must be visible to the engine host.

If you do not specify the `-t` option, the objects are exported to the current working directory.

If you specify the `-p` option, the project object and its root directory are exported.

If you specify the `-r` option and the object specified for export is a project or a directory, all of the objects used in the project or directory hierarchy are exported.

Options and arguments

`-delimiter` *`delimiter_value`*

Specifies the delimiter that separates the parts in an object's four-part name. Use this to export to a database that uses a different delimiter than the one used in the current database. For example, to export from a 4.4 to a 4.2.1 UNIX database, use a colon for *`delimiter_value`*. You must use the `-nid` option with this option.

The default is "@."

file_spec

Specifies the name of the file or directory to be exported.

-leave_wa_paths|-lwp

Preserves the current database's work area path for the exported objects.

When the objects are imported into a new database, their work area paths are already set to the old database's path.

-p|-project *project_spec*

Specifies the project being exported.

project_spec

Specifies the name of the project to be exported.

-q|-quiet

Suppresses messages output by the command.

-r|-recurse

Recursively exports all of the objects that are members of the directory or project hierarchy.

-t|-to *pathname*

Specifies the directory path to which the specified object will export. If you are exporting on a remote client (Windows), or local client (UNIX), you must use this option. The path you specify must be visible to the engine and must be writable by *ccm_root*.

Examples

Windows:
From the work area in the `c:\users\doug\cm_test_db\job-1\job` directory, you can do any of the following:

- Export the `src\foo.c`object to the `/users/doug/export_dir` directory.

  ```
  ccm export -to \\users\doug\export_dir src\foo.c
  ```

- Recursively export the `job-1` project to the `/users/doug/export_dir` directory.

  ```
  ccm export -to \\users\doug\export_dir /recurse -p job-1
  ```

- Export the `src` directory object to the `/users/doug/export_dir` directory.

  ```
  ccm export -to \\users\doug\export_dir src
  ```

- Export the `src` directory object, and every object under the directory, to the `/users/doug/export_dir` directory.

  ```
  ccm export -to \\users\doug\export_dir -recurse src
  ```

- Export the `src\foo.c-1` object version to the `/users/doug/export_dir` directory.

  ```
  ccm export -to \\users\doug\export_dir foo.c-1@csrc@1
  ```

UNIX:
From the work area in the /`users`/`doug/cm_test_db/job-1/job` directory, you can do any of the following:

- Export the `src/foo.c` object to the `/users/doug/export_dir` directory.

  ```
  ccm export -to /users/doug/export_dir src/foo.c
  ```

- Recursively export the `job-1` project to the `/users/doug/export_dir` directory.

  ```
  ccm export -to /users/doug/export_dir -recurse -p job-1
  ```

- Export the `src` directory object to the `/users/doug/export_dir` directory.

  ```
  ccm export -to /users/doug/export_dir src
  ```

- Export the `src` directory object, and every object under the directory, to the `/users/doug/export_dir` directory.

  ```
  ccm export -to /users/doug/export_dir -recurse src
  ```

- Export the `src`/`foo.c-1` object version to the `/users/doug/export_dir` directory.

  ```
  ccm export -to /users/doug/export_dir foo.c-1:csrc:1
  ```

Note The default delimiter is the `@` ("at") sign.

Caveat

You can run this command from a remote client (Windows) or local client (UNIX), but you can only export objects to paths visible to the engine process and writable by the *ccm_root* user.

Related topics

- import command

finduse command

Synopsis

```
ccm finduse [finduse_scope_spec] file_spec

ccm finduse -task [finduse_scope_spec] task_id [task_id...]

ccm finduse -fol|/folder [finduse_scope_spec] folder_id [folder_id...]

ccm finduse -baseline [baseline_spec]

ccm finduse [-q|-query "query_expression"]
            [-n|-name object_name] [-o|-owner owner]
            [-s|-state state] [-t|-type type]
            [-v|-version version] [-i|-instance instance]
            [finduse_scope_spec]  [-p project_spec]
```

Description and uses

The `finduse` command searches the database for uses of a specified object and returns a list of reference specifications identifying where it is used.

Options and arguments

`-apg|-all_project_groupings`

Finds uses of the object in all project groupings.

`-baseline` *`baseline_spec`*

Displays the project groupings that use the specified baseline. The default scope is `-all_project_groupings`, which shows all project groupings that use the specified baseline. A baseline is considered to be "used" in a project grouping If it is that project grouping's baseline. To view the baseline, use the project grouping's **Properties** dialog, or the `ccm project_grouping -show baseline` command.

The various scopes on project groupings restrict the project groupings that are returned in the usual way. These scopes are:

```
-all_project_groupings
-working_project_groupings
-my_project_groupings
-prep_project_groupings
-shared_project_groupings
```

file_spec

Specifies the name of the file, directory, or project for which the uses are to be found.

finduse_scope_spec

Specifies the scope of the search. The valid *finduse_scope_spec* values are:

```
[-working_proj]
[-shared_proj]
[-prep_proj]
[-released_proj]
[-all_proj]
[-personal_folder]
[-shared_folder]
[-prep_folder]
[-non_write_folder]
[-all_folder]
[-all_baseline|-all_baselines]
[-wpg|-working_project_groupings]
[-mpg|-my_project_groupings]
[-ppg|-prep_project_groupings]
[-spg|-shared_project_groupings]
[-apg|-all_project_groupings]
```

When a *finduse_scope_spec* is not specified, `-all_proj` is used.

`-f|-folder` *folder_id*

Finds all projects with the folder *folder_id* in their update properties.

`-i|-instance` *instance*

Finds all projects that include objects with the instance number *instance*.

`-mpg|-my_project_groupings`

Finds uses of the object in all private project groupings owned by the current user.

`-n|-name` *object_name*

Finds all projects that include objects with the name *object_name*.

`-o|-owner` *owner*

Finds all projects that include objects owned by *owner*.

`-ppg|-prep_project_groupings`

Finds uses of the object in all prep project groupings.

`-p` *`project_spec`*

Finds all projects that include *`project_spec`*.

`-q|-query "`*`query_expression`*`"`

Specifies a query expression that yields a set of objects, and then displays the uses of each object in the set. The `-q` option cannot be used with the `-p` option.

`-spg|-shared_project_groupings`

Finds uses of the object in all shared project groupings.

`-s|-state` *`state`*

Finds all projects that include objects in the state *`state`*.

`-t|-task` *`task_id`*

Finds all projects that use the task *`task_id`* in their update properties, either directly or through a folder. Also finds the folders that contain the task *`task_id`* if a folder scope is specified using the `finduse_scope_spec`.

`-t|-type` *`type`*

Finds all projects that include objects of type *`type`*.

`-v|-version` *`version`*

Finds all projects that include objects with the version *`version`*.

`-wpg|-working_project_groupings`

Finds uses of the object in all working project groupings.

Examples

- Find all uses of the object version named `display.c` in projects.

```
ccm finduse -name display.c
display.c-1 integrate epresley csrc ico_proj 1
ico_proj/src/display.c-1@ico_proj-1
display.c-2 integrate epresley csrc ico_proj 1
ico_proj/src/display.c-2@ico_proj-int2
display.c-3 integrate epresley csrc ico_proj 1
Object not used in scope
display.c-4 integrate epresley csrc ico_proj 1
ico_proj/src/display.c-4@ico_proj-epresley
```

- Find all uses in projects of the version of `draw.c` that is being used in the current directory.

```
ccm finduse draw.c
draw.c-1 integrate epresley csrc gditest EAP#3 EAP#274
    gditest/draw.c-1@gditest-org
    gditest/draw.c-1@gditest-shared2.1
    gditest/draw.c-1@gditest-visible2.1
    gditest/draw.c-1@gditest-int2.1
    gditest/draw.c-1@gditest-org1
```

- Find all folders containing task `128`.

```
ccm finduse -all_folders -task 128
Task EAP#128:  Correct color of icons
Folder EAP#3:  All Completed Tasks for Release 1.2
     Folder EAP#7:  bill's Completed Tasks for Release 1.2
     Folder EAP#8:  Integration Testing Tasks for Release 1.2
```

- Find all personal folders containing object `draw.c-2:csrc:EAP#1`.

```
ccm finduse -personal_folder draw.c-2:csrc:EAP#1
draw.c-2 integrate bill csrc draw_proj EAP#1 EAP#128
 Folder EAP#7:  bill's Completed Tasks for Release 1.2
     Folder EAP#9:  bill's Assigned or Completed Tasks for Release 1.2
```

- Find *prep* projects containing task `128`.

```
ccm finduse -prep_proj -task 128
Task EAP#128:  Correct color of icons
draw_proj-int1.2
util_proj-int1.2
```

- Find all projects that are using folder `7`.

```
ccm finduse -folder 7
Folder EAP#7:  bill's Completed Tasks for Release 1.2
draw_proj-bill
util_proj-bill
```

folder command

Synopsis

Compare Folders

```
ccm folder -comp|-compare folder_spec1
           -un|-union |
           -int|-intersection |
           -not|-not_in
           [-f|-format "format_string"] [-ns|-no_sort] [-u]
           folder_spec2
```

Copy Folder

```
ccm folder -cp|-copy folder_spec
           [-e|-existing existing_folder_spec [-append]] |
           [-new "new_folder_name"]
           [-y] [-q|-quiet]
```

Create Folder

```
ccm folder -cr|-create -n|-name "folder_name"
           [-us|-usable usable_by]
           [-w|-writable writable_by]
           [-qu|-query] [-q|-quiet]
           query_spec
```

Delete a Folder

```
ccm folder -delete [-y] folder_specs
           [-q|-quiet]
```

Find Uses of a Folder

```
ccm folder -fu|-find_use
           [-f|-format "format_string"] [-ns|-no_sort] [-u]
           folder_spec
```

List Folders

```
ccm folder -l|-list [scope]
           [-f|-format "format_string"] [-ns|-no_sort] [-u]
```

Modify Folders

```
ccm folder -m|-modify
           [-at|-add_task|-add_tasks task_specs] [-related]|
           [-rt|-remove_task|-remove_tasks task_specs] [-related]|
```

```
            [-up|-update] |
            [-mode {man|manual | uq|use_query}] |
            ["query_expression"] |
            [-n|-name "name_string"] |
            [-us|-usable usable_by] |
            [-w|-writable writable_by]
            [-y] [-q|-quiet]
            folder_specs
```

Show Folder Information

```
ccm folder -sh|-show
           i|info|information [-v|-verbose]|
           obj|objs|objects |
           t|task|tasks [-v|-verbose]
           [-f|-format "format_string"] [-ns|-no_sort] [-u]
           folder_specs
ccm folder -sh|-show
           mode |
           n|na|name |
           q|qu|query |
           u|us|usable |
           w|wr|writable
           folder_specs
```

Description and uses

Use the `folder` command to perform the following task-based Rational Synergy operations:

- Compare two folders
- Copy a folder
- Create a folder
- Delete one or more folders
- Show folder
- Modify folders
- Find the uses of a folder
- List folders

Options and arguments

`-at|-add_task|-add_tasks` *task_specs*

Adds one or more tasks to the specified folder.

You can use this option only with the -modify option, and the folder to which you are adding tasks must be writable by you.

-append

Appends the contents of the source folder (*folder_id1*) to the contents of the destination folder (*folder_id2*).

You can use the -append option with the -copy and -existing options only.

-comp|-compare *folder_spec1 folder_spec2*

Compares the contents of the specified folders. A selection set is populated with the tasks listed in the output.

Use the -f option to change the command's output format. Use -u to suppress automatic numbering of the output and -ns to suppress sorting.

The default output format for folder -compare is:

```
Task %displayname: %task_synopsis
```

where:

%displayname is %name if DCM is not enabled.
%displayname is <*database_ID*><*DCM_delimiter*><*task_number*> if DCM is enabled.
%task_synopsis is a description of the task.

The differences shown are relative to *folder_spec1*. The types of comparisons you can perform are:

— union (show the tasks that are in either of the folders)
— intersection (show the folders' common tasks)
— not_in (show the tasks that are in *folder_spec1* but not in *folder_spec2*)

If the command is successful, the return value is 0; otherwise, it is non-zero.

-cp|-copy

Copies all folder definitions from *folder_spec* to either a new folder (-new "*new_folder_name*") or an existing folder (-e|-existing *existing_folder_spec*). Additionally, you can use the -append option to append the contents of the source folder (*folder_id1*) to the contents of an existing destination folder (*folder_id2*).

Use this option with `-q` (quiet mode) to suppress the command's output, except the folder ID if a folder was created.

If the command is successful, the return value is `0`; otherwise, it is non-zero.

`-cr|-create`

Enables you to create a folder with the specified properties.

Use this option with `-q` (quiet mode) to suppress all of the command's output except the folder ID.

You must use this option with the `-name` option.

If the command is successful, the return value is `0`; otherwise, it is non-zero.

`-delete`

Deletes the specified folders. If you specify the `-y` option as well, Rational Synergy deletes the folders without displaying the confirmation message. (See the Caution for the `-y` option.)

The folder you are deleting must be writable by you.

Use this option with `-q` (quiet mode) to suppress the command's output except a count of all folders deleted.

If the command is successful, the return value is `0`; otherwise, it is non-zero.

`-e|-existing` *`existing_folder_spec`*

Specifies the existing folder to which you are copying tasks from *`folder_spec`*.

The copy operation sets the destination folder to add tasks manually.

You can use this option with the `-copy` option only, and the destination folder must be writable by you. Additionally, you can use the `-append` option with the `-copy` and `-`

`existing` options to append the contents of the source folder (*`folder_id1`*) to the contents of the destination folder (*`folder_id2`*).

`-f|-format "`*`format_string`*`"`

Specifies the command's output format. The default format depends on the other options you use with `-format` (that is, `-finduse`, `-list`, or `-show`) and those options' keyword arguments. See the options' descriptions for their default output formats.

The format can contain a combination of text and keywords. Keywords are replaced by specific information about each object as they are displayed. For example, the keyword `%owner` is replaced with `sue` if an object owned by user *sue* is displayed.

The name of any existing attribute can be used as a keyword. In addition, a number of built-in keywords are defined, such as `%displayname` and `%task_number`. See Built-In keywords for a list.

`-fu|-find_use`

Finds where the specified folder is in use in the current database.

Use the `-format` option to change the command's output format. Use `-u` to suppress automatic numbering of the output and `-no_sort` to suppress sorting.

The default output format for `folder -finduse` is:

```
%name %status %owner %version
```

where:

- `%name` is the project name.
- `%status` is the project status.
- `%owner` is the owner of the project.
- `%version` is the project version.

`folder_spec`

Specifies the ID of the folder that you are listing, adding, removing, or changing. Folder specs can be separated by a comma or white space.

For this argument's syntax, see Folder specification.

You must use this option with the `folder_spec` option.

`-int|-intersection`

Indicates that the compare results will show all tasks that are common to both of the specified folders.

You can use this option only with the `-compare` option.

`-l|-list [scope]`

Lists all folders, or the folders specified by *`scope`*.

Use the `-format` option to change the command's output format. Use `-u` to suppress automatic numbering of the output, and `-no_sort` to suppress sorting.

The default output format for `folder -list` is:

`Folder %displayname: %description`

where:

`%displayname` is `%name` if DCM is not enabled.
`database_ID` is *`<database_ID><DCM_delimiter><task_number>`* if DCM is enabled.
`%description` is the folder name.

The *`scope`* argument must have one of the following values:

- `all_personal`
- `all_build_mgrs`
- `all_shared`
- `all_non_writable`
- `all`

By default, all of your personal folders are listed.

If the command is successful, the return value is `0`; otherwise, it is non-zero.

`-mode {man|manual | uq|use_query}`

Defines the folder's contents either manually (`manual`) or using a query (`use_query`).

You can use this option only with the `-modify` option.

Rational Synergy treats changes in mode from manual to query-based in the following ways.

- If you change a manual folder to a query-based folder and the folder was previously query-based, its last query is used.
- If you change a manual folder to a query-based folder and the folder was never query-based and there is a user-defined query (default_task_query), the user-defined query becomes the query.
- If you change a manual folder to a query-based folder and the folder was never query-based and there is **not** a user-defined query (default_task_query) and you are working as the build manager, the query becomes `All Completed Tasks`.
- If you change a manual folder to a query-based folder and the folder was never query-based and there is **not** a user-defined query (default_task_query) and you are not working as the build manager, the query becomes `All Tasks Assigned to` *`your_user_name`*.

`-m|-modify`

Enables you to change a folder property by using any combination of the following sub options:

```
-at|-add_task task_specs [-related]
-rt|-remove_task|-remove_tasks task_specs [-related]
-up|-update
-mode {man|manual | uq|use_query}
query_spec
-n|-name "name_string"
-us|-usable usable_by
-w|-writable writable_by
-q|-quiet
folder_specs
```

The `-modify` option accepts multiple sub options.

You can use this option to make incremental changes to the folder's default-style or custom-style query.

If folder A uses a default-style query of `All completed tasks for release 2.1`, and you perform a `ccm folder -modify -release 2.2` command on it, the query will incrementally change to `All completed tasks for release 2.2`.

If folder A uses a custom-style query and you perform a `ccm folder -modify` *`query_spec`* command, the query is replaced. Additionally, if you do not use the `-custom` option, but you use the `-task_scope` option, the folder's query will change to default-style.

The folder on which you are changing the option must be writable by you.

Use this option with `-quiet` to suppress the command's output. Use the `-y` option to suppress the confirmation message.

If you use the `-add_task` option to add an excluded task to a folder, you will receive a confirmation message. Answer appropriately to continue.

If the command is successful, the return value is `0`; otherwise, it is non-zero.

`-n|-name "`*`folder_name`*`"`

Renames the specified folder.

You can use this option only with the `-create` or `-modify` option, and the folder you are renaming must be writable by you.

`-new "`*`new_folder_name`*`"`

Specifies the name of the new folder to which you are copying folder properties from *`folder_spec`*.

The copy operation sets up the destination folder, *`new_folder_name`*, to add tasks manually.

You can use this option only with the `-cp` option.

`-not|-not_in`

Indicates that the compare results will show all tasks that are contained in *`folder_spec1`* that are not contained in *`folder_spec2`*.

You can use this option only with the `-compare` option.

`-ns|-no_sort`

Specifies to not sort the command's output.

`-q|-quiet`

Executes the command with a reduced number of output messages. When you create a folder, this option causes the output to contain only the folder ID (for example, `3`).

`-qu|-query`

Makes a folder query-based.

You can use this option only with the `-create` option.

The query is defined according to the following conditions if you are using no *`query_spec`*:

- If there is a user-defined query (default_task_query), the user-defined query becomes the query.
- If there is **not** a user-defined query (default_task_query) and you are working as the build manager, the query becomes `All Completed Tasks`.
- If there is **not** a user-defined query (default_task_query) and you are not working as the build manager, the query becomes `All Tasks Assigned to` *`your_user_name`*.

`query_spec`

Changes the query that is used to add tasks to the specified folders.

The syntax for *`query_spec`* is:

```
[-cus|-custom "query_expression"]
[-db|-dbid|-database_id database_id]
[-plat|-platform platform]
[-rel|-release release]
[-sub|-subsystem subsystem]
[-ts|-scope|-task_scope task_scope]
```

where *`task_scope`* is one of the following folders:

`user_defined`
`all_my_assigned_or_completed` (default)
`all_my_assigned`
`all_my_completed`

```
all_my_tasks
all_completed
all_tasks
```

Any user-created queries in the CLI are saved as custom queries (with no query clauses); the scope associated with the specified option is not saved.

If you do not specify a task scope, Rational Synergy uses the value `all_my_assigned_or_completed`.

`query_expression` is the same as the *`query_expression`* for the query command.

You can use this argument only with the `-create` or `-modify` option.

`-related`

Use `-related` with `-add_tasks` or `-remove_tasks` to relate all tasks that have a status of *completed* to the specified folder or folders.

`-rt|-remove_task` *`task_specs`*

Removes one or more tasks from the specified folder. See Task specification for the syntax of the *`task_specs`* argument.

You can use this option only with the `-modify` option, and the folder you are modifying must be writable by you.

`scope_spec`

Changes the query that is used to list folders.

`scope_spec` is one of the following:

```
all_personal | all_build_mgrs | all_shared | all_non_writable | all
```

`-sh|-show`

Shows the properties of the specified folders. When you use this option with the `info`, `objs`, or `tasks` keywords or their variants, a selection set is populated with the objects or tasks listed in the output.

Use the -format option to change the command's output format. Use -u to suppress automatic numbering of the output, and -no_sort to suppress sorting.

If you are using one of the following keywords, the -format option is allowed:

```
i|info|information
obj|objs|objects
t|task|tasks
```

If you are using -show info, you can use the -verbose option to display all folders with all associated tasks and all objects that are affected by tasks. If you are using -show tasks, you can use the -verbose option to display all tasks for each folder and all objects that are affected by tasks.

The default output format for folder -show information is:

```
Folder %displayname: %description
```

where:

%displayname is %name if DCM is not enabled.
%displayname is <*database_ID*><*DCM_delimiter*><*task_number*> if DCM is enabled.
%description is the name of the folder.

These lines are followed by additional information on some of the folder's properties.

The default output format for folder -show objects is:

```
%objectname %status %owner %task
```

where:

%objectname is the object's *name-version:type:instance*.
%status is the status of the object.
%owner is the owner of the object.
%task is the task associated with the object.

The default output format for folder -show tasks is:

```
Task %displayname: %task_synopsis
```

where:

`%displayname` is `%name` if DCM is not enabled.

`%displayname` is `<database_ID><DCM_delimiter><task_number>` if DCM is enabled.

`%task_synopsis` is a description of the task.

You can also show folder properties using one of the following keywords if you are **not** using the `-format` option:

```
mode
n|na|name
q|qu|query
u|us|usable
w|wr|writable
```

If the command is successful, the return value is `0`; otherwise, it is non-zero.

task_specs

Specifies the IDs of the tasks. For this argument's syntax, see Task specification.

`-u`

Suppresses automatic numbering of this command's output ("un-numbered").

`-un|-union`

Indicates that the compare results will show all tasks that are in either of the specified folders.

You can use this option only with the `-compare` option.

`-up|-update`

Updates the objects associated with the specified folder's tasks.

You can use this option only with the `-modify` option.

`-us|-usable` *usable_by*

Makes the folder usable by `Owner`, `Build_Manager`, `All`, or `None`.

You can use this option only with the `-create` or `-modify` option, and the folder you are modifying must be writable by you.

The default output format for `folder -usable` is:

```
%owner
```

where:

— `%owner` is the owner of the project.

`-v|-verbose`

If you are using `-show info`, you can use the `-verbose` option to display all folders with all associated tasks and all objects that are affected by tasks. If you are using `-show tasks`, you can use the `-verbose` option to display all tasks for each folder and all objects that are affected by tasks.

`-w|-writable` *writable_by*

Makes the folder writable by `Owner`, `Build_Manager`, `All`, or `None`. If you attempt to set this option to `None`, Rational Synergy displays a confirmation message, since this action will make the folder read-only for all users. You can use the `-y` option to suppress the confirmation message.

You can use this option only with the `-create` or `-modify` option, and the folder you are modifying must be writable by you.

The default output format for `folder -writable` is:

```
%owner
```

where:

— `%owner` is the owner of the project.

`-y`

Suppresses the confirmation message that would otherwise be displayed when you delete or modify a folder. You can use this option only with the `-delete` or `-modify` options.

Confirmation messages provide important safeguards against user error. Use the `-y` option with caution.

Related topics

- folder Examples

folder Examples

View examples for the following operations:

- Compare Folders
- Copy Folders
- Create a Folder
- Delete a Folder
- Find Uses of a Folder
- List Folders
- Modify Folders
- Rename a Folder
- Show Folder Information

Compare Folders

- Show the tasks that are in either folder 154 or folder 155.

```
ccm folder -comp 154 -un 155
1) Task 12: System error when time zone changes
2) Task 15: Correct spelling errors in output
3) Task 19: Rewrite messaging module
4) Task 26: Close box no longer active
5) Task 31: Wrong window receives message
6) Task 40: Auto-calculation gives incorrect result
7) Task 53: Download of images occurs too slowly
```

- Show the tasks that folders 154 and 155 have in common.

```
ccm folder -comp 154 -int 155
1) Task 15: Correct spelling errors in output
2) Task 19: Rewrite messaging module
3) Task 26: Close box no longer active
4) Task 40: Auto-calculation gives incorrect result
```

- Show the tasks that are in folder 154 but not in folder 155.

```
ccm folder -comp 154 -not 155
1) Task 12: System error when time zone changes
2) Task 31: Wrong window receives message
```

- Compare two folders in the database. First, query for all folders.

```
ccm folder -list all
```

  Output similar to the following is displayed.

```
1) Folder 27: All completed tasks for release 2.0
2) Folder 32: All tested tasks for release 2.0
3) Folder 13: Bill's tasks for release 2.0
```

Then, show tasks in the `All completed tasks for release 2.0` folder that are not in the `All tested tasks for release 2.0` folder.

```
$ ccm folder -compare @1 -not_in @2
Task 304: Change splash screen for release 2.0
Task 306: Change copyright for release 2.0
```

Copy Folders

- Copy folder 95 to a new folder named `Tasks Completed for Release 3.4 on September 15, 1997`.

  ```
  ccm folder -copy 95 -new "Tasks Completed for Release 3.4 on September
  15, 1997"
  Folder '95: Tasks Completed for Release 3.4' copied to '158: Tasks
  Completed for Release 3.4 on September 15, 1997'
  ```

- Copy folder 95 to an existing folder, number 103.

  ```
  ccm folder -cp 95 -existing 103
  Folder '95: Tasks Completed for Release 3.4' copied to '103:  Tested
  Tasks for Release 3.4"
  ```

- Copy folder `All completed tasks for 2.1` to an existing folder, `All completed tasks for 2.0`, merging the two folders' contents.

  ```
  ccm folder -copy All completed tasks for 2.1 -existing All completed
  tasks for 2.0 -append
  ```

Create a Folder

- Create a new folder named `Tested Tasks for Release 3.5` that is writable by its owner and usable by all, and suppress all output from the command except for the folder ID.

  ```
  ccm folder -cr -n "Tested Tasks for Release 3.5" -w Owner -us All -q
  159
  ```

- Create a new folder named `My Tasks for Release 3.5` that uses a *`task_spec`* and a *`release`* value for a *`query_spec`*.

  ```
  ccm folder -cr -name "My Tasks for Release 3.5" -ts all_my_tasks -rel
  3.5
  Created folder 160.
  ```

Delete a Folder

- Delete folders 109,110, and 158.

```
ccm folder -delete 109-110,158
Are you sure that you want to delete folder '109: Tasks Completed for
Release 2.1 on May 1, 1996'? (Yes/All/No) [No] y
Deleted folder '109: Tasks Completed for Release 2.1 on May 1, 1996'.
Are you sure that you want to delete folder '110: Tasks Completed for
Release 2.2 on July 1, 1996'? (Yes/All/No) [No] y
Deleted folder '110: Tasks Completed for Release 2.2 on July 1, 1996'.
Are you sure that you want to delete folder '158: Tasks Completed for
Release 3.4 on September 15, 1997'? (Yes/All/No) [No] y
Deleted folder '158: Tasks Completed for Release 3.4 on September 15,
1997'.
```

Find Uses of a Folder

- Find where the folder named `All Completed Tasks for Release 2.1` is used in the current database.

```
ccm folder -list all_non_writable -format "%displayname %description"
All Non-Writable Folders
1) 42 All Completed Tasks for Release 2.1
2) 89 All Completed Tasks for Release 2.2
```

List Folders

- List all of the build manager's folders in the current database.

  ```
  ccm folder -list all_build_mgrs
  1) Folder 42: All Completed Tasks for Release 2.1
  2) Folder 95: Tasks Completed for Release 3.4
  ```

- List all of your personal folders.

  ```
  ccm folder -list
  1) Folder 111: mary's Insulated Development Folder
  2) Folder 145: mary's Completed Tasks for Release 4.2
  3) Folder 146: mary's Assigned Tasks
  ```

- List all folder templates in the database.

  ```
  ccm folder -list -template all
  ```

Modify Folders

- Add tasks 5-9 to folder 95.

  ```
  ccm folder -modify -at 5-9 95
  Updating folder 95: Tested Tasks for Release 3.2 ...
     Added task 5
     Added task 6
  ```

```
   Added task 7
   Added task 8
   Task 9 is already in the folder
 Added 4 tasks.
```

- Remove tasks 5-9 from folder 95.

```
ccm folder -modify -rt 5-9 95
Updating folder 95: Tested Tasks for Release 3.2 ...
   Removed task 5
   Removed task 6
   Removed task 7
   Removed task 8
   Removed task 9
Removed 5 tasks.
```

- Add multiple tasks (5, 12, 14) to folder 51.

```
ccm folder -modify -add_task 5,12,14 51
```

- Update the contents of folder 160.

```
ccm folder -m -up 160
Updated folder '160: My Tasks for Release 3.5'.
```

- Change the mode of folder 111 so that it uses a query to add tasks.

```
ccm folder -modify -mode use_query 111
Folder '111: mary's Insulated Development Folder' has been changed to
add tasks using a query.
```

- Change folder 111 so that it uses the `all_my_tasks` scope and release 3.5 to add tasks.

```
ccm folder -modify -ts all_my_tasks -rel 3.5 111
The query for folder '111: mary's Insulated Development Folder' has
been changed to: owner='mary' and release='3.5'
```

- Change the use and write permissions to *`use_permission`* and *`write_permission`*, respectively, for *`folder_number`*.

```
ccm folder -modify -usable use_permission -writable write_permission
folder_number
```

`read_permission` and *`write_permission`* can have any of the following values:

- Owner
- Build_Manager
- All
- None

> **Caution** Changing the use or write permission to None makes the folder unusable or read-only, respectively, by all users.

Rename a Folder

- Change the name of *folder_number*.

```
ccm folder -modify -name "new_folder_name_string" folder_number
```

> **Note** If the folder is controlled by a template, you will receive, and must respond to, a confirmation message when you change the folder's name. You can suppress the message by adding the -y option to the command.

Show Folder Information

- Show folder 160 information.

```
ccm folder -sh info 160
Folder '160: My Tasks for Release 3.5'
    Owner:        mary
    Writable By:  Owner
    Usable By:    Owner
    Query Type:   All My Tasks
    Query:        owner='john' and release='3.5'
```

- Show the tasks in folder 111.

```
ccm folder -show tasks 111
1) Task 19: Rewrite messaging module
2) Task 26: Close box no longer active
3) Task 31: Wrong window receives message
4) Task 40: Auto-calculation gives incorrect result
5) Task 53: Download of images occurs too slowly
```

- Show the objects that are associated with folder 160.

```
ccm folder -sh objects 160
1) UTIL.C-2:csrc:1    integrate  mary 19
2) MSGS.C-3:csrc:1    integrate  mary 19
3) MSGS.H-2:incl:1    integrate  mary 19
4) DIALOG.C-8:csrc:1  integrate  mary 57
5) DIALOG.H-13:incl:1 integrate  mary 57
```

folder_template command

Synopses

Create a Folder Template

```
ccm ft|folder_temp|folder_template -c|-create
          [-desc|-description description]
          [-us|-usable usable_by] [-w|-writable writable_by]
          [-mode {man|manual | uq|use_query]
          [-task_scope|-ts task_scope]
          [-database_id|-dbid|-db dbid]
          [-must_be_local|-nomust_be_local]
          [-release|-rel release]
          [-platform|-plat platform]
          [-subsystem|-sub task_subsystem]
          [-custom "query_expression"]
          "folder_template_name"
```

Modify Folder Templates

```
ccm ft|folder_temp|folder_template -m|-modify
          [-desc|-description description]
          [-us|-usable usable_by] [-w|-writable writable_by]
          [-mode {man|manual | uq|use_query]
          [-task_scope|-ts task_scope]
          [-database_id|-dbid|-db dbid]
          [-must_be_local|-nomust_be_local]
          [-release|-rel release]
          [-platform|-plat platform]
          [-subsystem|-sub task_subsystem]
          [-custom "query_expression"]
          folder_template_specs [folder_template_specs]*
```

Delete Folder Templates

```
ccm ft|folder_temp|folder_template -d|-delete
          folder_template_specs [folder_template_specs]*
```

List Folder Templates

```
ccm ft|folder_temp|folder_template -l|-list
```

Set Controlling Database for Folder Templates

```
ccm ft|folder_temp|folder_template -cdb|-controlling_database
          (-local | -handover database_id | -accept database_id)
```

Show Detailed Properties of Folder Templates

```
ccm ft|folder_temp|folder_template -sh|-show (i|info|information)
          folder_template_specs [folder_template_specs]*
```

Show Specific Property of Folder Templates

```
ccm ft|folder_temp|folder_template -sh|-show
          (description|mode|description|query|(u|us|usable)|(w|wr|writable))
          folder_template_specs [folder_template_specs]*
```

Description and uses

Folder templates provide a pattern used to create folders. The process rule can use folder templates as part of the basis for how a project is updated (reconfigured).

Use the `folder_temp` command to perform the following task-based Rational Synergy operations:

- Create a folder template
- Modify folder templates
- Delete folder templates
- List folder templates
- Set controlling database for folder template(s)
- Show detailed properties of folder templates
- Show specific properties of folder templates

Options and arguments

`-accept`

Specifies that the object is set to accept control from the specified database.

`-cdb|-controlling_database`

Specifies a controlling database for the specified folder templates.

`-cr|-create`

Enables you to create a folder template and give it the specified *`folder_template_name`*. If a folder template that has that name already exists, the command reports an error and the new folder template is not created.

If the command is successful, the return value is `0`; otherwise, it is non-zero.

`-delete`

Deletes one or more specified folder templates. If a specified folder template is being used by a process rule or if it is a system-defined folder template, the command reports an error and the folder template is not deleted.

A folder template you are deleting must be writable by you.

If the command is successful, the return value is `0`; otherwise, it is non-zero.

`-dbid|-database_id` *`dbid`*

Specifies the database ID that is associated with the folder template you are creating or modifying. If you do not specify a *`dbid`*, the database ID of the current database is used.

`-desc|-description` *`description`*

Specifies a string that is used after keyword expansion when creating folders from the folder template. If you do not specify a description, the folder template name is used as a default value.

You can give a folder template any description, using any character, except the pipe (|) character. The folder template description can include the following three keywords in any combination: `%owner, %release, %database`. A folder template's description does not have to include keywords. When the description of a folder template does

not contain a keyword, all folders created from this folder template will have the same description.

For example, if you create a folder template whose description is `Completed Tasks for Release %release`, the keyword `%release` is expanded to the release value of the projects that are using a process rule containing this folder template. The keyword `%release` is expanded when this folder template creates a folder. For instance, when a project with release `2.0` uses a process rule that contains a folder template whose description is `Completed Tasks for Release %release`, the folder created from this template and added to the project's update properties will have a description of `Completed Tasks for Release 2.0`.

The `%owner` keyword expands to the owner of the project whose update properties contain folders created from a folder template. For example, if a project owned by `joe` with a release of `3.1` uses a process rule that contains a folder template whose description is `%owner's Completed Tasks for Release %release`, a folder whose description is `joe's Completed Tasks for Release 3.1` will be created from this folder template and added to the project's update properties.

The `%database` keyword expands to the DCM database identifier of the database in which the project that is using the folder that is being created was created. For example, if a project owned by `joe` with a release of `3.1` is in a DCM database called `Bristol`, and is using a process rule that contains a folder template whose description is `%owner's Completed Tasks for Release %release from Database %database`, a folder whose description is `joe's Completed Tasks for Release 3.1 from Database Bristol` will be created from this folder template and added to the project's update properties.

folder_template_name

Specifies the name of the folder template or templates that you are creating or modifying. The folder template name is required and must be unique. Double quotes and single quotes are restricted in folder template names. For example, a folder template name of "%owner's tasks" is invalid.

folder_template_specs

Specifies the folder templates that you are creating or modifying.

Arguments of type *folder_template_specs* may be any of the following:

- A folder template name.
- A list of folder template names separated by comma.
- A four-part objectname.

- A selection set reference (@n or @), or a series of such references separated by commas.
- A cvid form (@=*n*), or a series of such separated by commas.
- A filename that contains one or more folder template names, each name placed on a separate line.

`-handover`

Specifies that control of the object is handed over from the current database to the specified database.

`-l|-list`

Lists all currently-defined folder templates.

`-local`

Specifies that local control is set, which breaks any previous DCM replication from another database.

`-mode {man|manual | uq|use_query}`

Defines the folder template's contents either manually (`manual`) or using a query (`use_query`).

You can use this option only with the `-create` or `-modify` option.

Rational Synergy treats changes in mode from manual to query-based in the following ways.

- If you change a manual folder template to a query-based folder template and the folder template was previously query-based, its last query is used.
- If you change a manual folder template to a query-based folder template and the folder template was never query-based and there is a user-defined query (default_task_query), the user-defined query becomes the query.
- If you change a manual folder template to a query-based folder template and the folder template was never query-based and there is **not** a user-defined query (default_task_query) and you are working as the build manager, the query becomes `All Completed Tasks`.
- If you change a manual folder template to a query-based folder template and the folder template was never query-based and there is **not** a user-defined query (default_task_query) and you are not working as the build manager, the query becomes `All Tasks Assigned to` *`your_user_name`*.

`-m|-modify`

Enables you to change a folder template property. The `-modify` option accepts multiple sub options. You can use this option to make incremental changes to the folder template's default-style or custom-style query.

The folder template on which you are changing the option must be writable by you.

If the command is successful, the return value is `0`; otherwise, it is non-zero.

`-must_be_local`

Specifies that a local folder must be used by the folder template for update properties of projects created locally.

The `-must_be_local` and `-nomust_be_local` options cannot be used together. If neither is specified when a folder template is created, the default is `-nomust_be_local`.

`-nomust_be_local`

Specifies that a non-local folder can be used by the folder template for update properties of projects created locally.

The `-must_be_local` and `-nomust_be_local` options cannot be used together. If neither is specified when a folder template is created, the default is `-nomust_be_local`.

`-plat|-platform` *platform*

Specifies the platform to which the change associated with the folder template applies. The platform choices are defined in the *CCM_HOME*`\etc\om_hosts.cfg` file (Windows), or `$CCM_HOME/etc/om_hosts.cfg` file (UNIX). If a folder template applies to multiple platforms, you should not set a platform value.

You can use this option only with the `-create` or `-modify` option.

`-qu|-query`

Makes a folder template query-based.

You can use this option only with the `-create` option.

Any user-created queries in the CLI are saved as custom queries (with no query clauses); the scope associated with the specified option is not saved.

`-r|-release` *`release_value`*

Specifies a release value for the folder template. Possible release values include, but are not limited to, the defined active release values in this database.

`-rt|-remove_task` *`task_specs`*

Removes one or more tasks from the specified folder. See Task specification for the syntax of the *`task_specs`* argument.

You can use this option only with the `-modify` option, and the folder you are modifying must be writable by you.

`scope_spec`

Changes the query that is used to list folders.

`scope_spec` is one of the following:

```
all_personal | all_build_mgrs | all_shared | all_non_writable | all
```

`-sh|-show`

Shows the properties of the specified folder templates. When you use this option with the `info` keyword, a selection set is populated with the objects or tasks listed in the output.

The default output format for `folder_template -show information` is:

```
Folder_Template %displayname: %description
```

where:

`%`*`displayname`* is `%name` if DCM is not enabled.

`%`*`displayname`* is `<`*`database_ID`*`><`*`DCM_delimiter`*`><`*`task_number`*`>` if DCM is enabled.

`%description` is the name of the folder.

These lines are followed by additional information on some of the folder's properties.

The default output format for `folder_template -show objects` is:

```
%objectname %status %owner %task
```

where:

`%objectname` is the object's *`name-version:type:instance`*.
`%status` is the status of the object.
`%owner` is the owner of the object.
`%task` is the task associated with the object.

If the command is successful, the return value is `0`; otherwise, it is non-zero.

`-sub|-subsystem` *`task_subsystem`*

Specifies the subsystem to which the task belongs (for example, Any, GUI code, CLI code, or documentation). If the subsystem specification contains spaces, you must enclose it in quotes.

You can use this option only with the `-create` or `-modify` option.

`-task_scope|-ts` *`task_scope`*

Specifies the task scope, where where *`task_scope`* is one of the following folders:

`all_owners_assigned_or_completed` (default)
`all_owners_assigned`
`all_owners_completed`
`all_owners_tasks`
`all_completed`
`all_tasks`

`-us|-usable` *`usable_by`*

Makes the folder template usable by `Owner`, `Build_Manager`, `All`, or `None`.

You can use this option only with the `-create`, `-modify`, or `-controlling_database` options, and the folder template must be writable by you.

The default output format for `folder_template -usable` is:

```
%owner
```

where `%owner` is the owner of the project.

`-w|-writable` *writable_by*

Makes the folder writable by `Owner`, `Build_Manager`, `All`, or `None`. If you attempt to set this option to `None`, Rational Synergy displays a confirmation message, because this action will make the folder template read-only for all users.

You can use this option only with the `-create` or `-modify` option, and the folder template you are modifying must be writable by you.

The default output format for `folder -writable` is:

```
%owner
```

where `%owner` is the owner of the project.

Examples

- View all personal folder templates.

  ```
  ccm folder_template -list -template all_personal
  ```

- Create a folder template whose description is "`%owner's Completed Tasks for Release %release from Database X`", set the folder template to use a query, and enter a folder query. You do not need to set who can write and use the folder template because the default setting is `owner`.

  ```
  ccm folder_template -create -description "%owner's Completed Tasks for
  Release %release from Database X" -task_scope all_owners_completed
  -release "%release" -database_id X "Tasks completed by %owner for
  Release %release from Database X"
  ```

- Hand over control of a locally-controlled folder template to a database whose ID is A1.

  ```
  ccm folder_template -controlling_database -handover A1
  folder_template_spec
  ```

- In a DCM database, change folder template 99 so that it is not database-specific.

  ```
  ccm folder_template -modify -desc "Completed Tasks for Release
  %release" -database_id Any T99
  ```

- Modify a folder template with a name of `T99` to use a custom query to gather tasks that were completed before April 29, 2006.

  ```
  ccm folder_template -modify -custom "(status='completed') and
  (completion_date<time('4/29/06')) T99
  ```

- Do the following to define a default query that a folder template will use to populate its folders with tasks:

1. Set the scope.
2. Set the release.

Set this attribute for parallel development and folder template management reasons.

3. Set the subsystem, if necessary.
4. Set the platform, if necessary.

If a folder applies to multiple platforms, you do not need to set the platform value.

5. Set the database, if it is initialized to use DCM.

```
ccm folder_template -create -desc name -task_scope scope -release
%release -usable usable_by -writable writable_by
```

For example, create a new folder template. Folders created from this template will collect all completed tasks for the current release, and will be writable and usable by build managers.

```
ccm folder_template -create -desc "All Completed Tasks for Release
%release" -task_scope all_completed -release "%" -usable Build_Manager
-writable Build_Manager
```

Related topics

- folder command

fs_check command

Synopsis

```
ccm fs_check [-d|-dir directory_path] [-f|-fix] [object_spec...]
             [-v|-verbose][-t|-type type] [-e|-empty_skip] [-u|-unused_skip]
             [-nd|-noduplicates] [-w|-windows]
             [-n|-null_byte][-z|-zero_counts]
```

Description and uses

Use the `ccm fs_check` command to check the consistency of a Rational Synergy database's file systems. By default, the `ccm fs_check` command checks that:

- Every file in the cache area corresponds to an existing object version.
- Every file in the archive area corresponds to one or more static object versions.
- Every entry in an archive file corresponds to one static object version.
- The source for a project or directory is empty.

Checking all the files in the cache and archive areas takes time and memory resources, and may be suppressed using the `-u|-unused` option.

To ensure consistency, execute `ccm fs_check` to check your entire database. This command can be used regularly to reduce the disk space taken up by cache files. However, as the check can take a long time on large databases, you can perform a quicker check by checking only specific types of objects. You can use the `-t` option to check only objects of the specified type, or you can check a list of objects using *`object_specs`* (for example, using query results). You cannot use both the `-t` option and a list of objects. In order to examine the results, you should direct the output to a file.

If unexpected or extra files or archive entries are found, they are reported individually and summarized at the end. However, such cases are not counted as errors and do not cause `ccm fs_check` to fail with a non-zero exit status. The `-fix` option to `ccm fs_check` does not remove these extra entries; doing so might lead to data loss in cases where you have created such files manually for your own purposes, or where you have restored a file system and metadata backup taken at slightly different times. Contact Rational Technical Support for assistance in removing unwanted extra cache and archive entries.

All users can perform this operation, however, you must be in the *ccm_admin* role to perform the `-fix` option.

Options and arguments

`-d|-dir` *`directory_path`*

Specifies the directory into which inconsistent archive entries are written. By default, these files are written to *`database_path`*`/st_root/tmp/check`.

`-e|-empty_skip`

Suppresses warnings about non-empty sources on projects and directories.

`-f|-fix`

Fixes some simple errors, including the following:

- If you unpack a database from a pack file created on UNIX, it is likely that cache files are in UNIX format. If this newline style is the only difference between the cache and archive, the `-f|-fix` flag causes the cache file to be deleted.
- If the cache file is zero length, but the archived content is not, the `-f|-fix` flag causes the cache file to be deleted.
- If the cache file has the wrong modify time, but is equal in content to the archive, use the `-f|-fix` option to update the modify time to be equal to the `source_modify_time` attribute.

`-nd|-noduplicates`

Specifies to not check for duplicate archive entries.Use this option to reduce the memory resources used when checking very large databases that might otherwise fail due to lack of memory. Since it reduces the strength of archive checking, it should only be used when necessary.

`-n|-null_byte`

Checks the source attribute for null (0x00) bytes. Generates a warning message when objects of type ascii and subtypes of ascii contain null bytes.

`object_spec`

Provides a list of objects to check. You can use query results for this argument.

You cannot use this argument with the `-t` option.

`-t|-type` *`type`*

Specifies the type of objects to check.

You cannot use this option with *`object_spec`*.

`-u|-unused_skip`

Suppresses the check that all cache and archive files and entries are used.

`-v|-verbose`

Generates more detailed information about each error. The errors report the following:

- Objects with no `source` attributes, excluding problems and tasks. These objects have no cache or archive entries, and are skipped.
- Files still archived by the old pre-4.1 archivers (SCCS, compress, and RCS, not `ccm_rcs`). This means that you must perform archive conversion.
- Objects with no cache files. Such objects were probably affected by an earlier execution of `ccm clean_cache`.
- Objects with no `source_modify_time` attributes. This is a minor error. Such objects have not been upgraded correctly to current database standards. You can create the `source_modify_time` attribute, of type `time`, and set it to the correct time (the time when the source file was last edited, before it was checked in). This should be the modify time on the cache file.
- Object cache files with times earlier than their `source_modify_time`s. This error is not serious, and might have been caused by the failure of a call to set the cache file time. Fix this by deleting the cache file **after** you have ensured that the archive entry is correct.

`-w|-windows`

Suppresses a warning message given when files differ only in carriage return characters.

`-z|-zero_counts`

In the summary at the end, zero counts (reporting conditions not seen) are normally suppressed. If you use the `-z` option, all counts are printed, including those with zero values. This option is useful when the output of `fs_check` is being analyzed by another program. It also allows you to detect when new checks have been added to the command.

> **Note** This option re-uses the `-z` option that previously meant skip the empty file check. If you previously used this option in your scripts, you must change your scripts.

Example

Check the file system consistency of the `project1` database and provide detailed output information.

Windows:

```
ccm fs_check -d c:\data\ccmdb\project1 -v
```

UNIX:

```
ccm fs_check -d /vol/hydra1/ccmdb/project1 -v
```

Related topics

- lmgr_status command

groups command

Synopsis

```
ccm groups    [-a|-assign] object_spec
              [-a|-assign] [-v|-value groupname] object_spec
              [-l|-list]
              [-c|-create] group_name
              [-e|-edit] group_name
```

Description and uses

A Rational Synergy database can contain many different collections of objects. It may not always be appropriate to allow all users with the applicable role to view, check out, and modify all objects. Group security allows restriction of check out and modify permissions to a specified group of users. In addition, read security, which limits visibility of objects to designated groups, can be specified. Use the `groups` command to implement and define security for objects.

If you are working as the group manager, group security allows you to:

- define a named group of users
- define and modify the users that are members of that named group
- restrict check out, read, and modify access of an object to specified named groups by assigning one or more groups to it.

A user working in a role that is allowed to create objects, such as *developer*, *writer*, *component_developer* or *build_mgr,* can restrict access of an object to specified groups if that object is the only version of that object and the user can modify that object.

Read security is implemented by providing access control to an object's source attribute. Users can query for objects and see other attributes regardless of any read restrictions. Read security applies to source objects which can be versioned, and does not apply to directories and projects.

> **Note** If you apply read security to an object that is currently in users' work areas, the files will still be readable by the users.

Three different levels of read access security can be defined:

- An object that has no read access restrictions to its source can be accessed by any user.
- An object that has one or more groups defined for read access will only allow access to the source if the user is a member of at least one of those groups. All other users will be denied access to the source contents of that object.

- An object with the highest level of security (no access to the source) can't be viewed, checked out, or modified, but other attributes can be viewed. However, users working in the *ccm_admin* role can always view the source contents of files.

Any object that is checked out inherits the same group security restrictions as its predecessor, including read security restrictions.

The following examples illustrate how security is applied and used for an object:

1. When no groups exist, or no groups are assigned to an object, there are no restrictions, meaning everyone can view, check out, and modify source files.
2. When one or more groups are created, and one group is assigned to that object, only users in the specified group can view, check out. and modify files. Users not in the specified group can only view the source objects. In other words, check out and modify security is implemented, but read security does not yet exist.
3. When one or more groups are created, and one group is given read security access (the ability to view source files), then all other groups do not have read access to the files. So once you start using the read security option, access to source contents is denied by default.

Note that read security can only be used with copy-based work areas. For additional information about setting up databases for read security, see the appropriate *Rational Synergy Administration Guide*.

If you have a directory in the public state that uses group security and the user is not a member of any of the directory's groups, the user is still allowed to create new objects, add them to, or unuse them from the directory. For best results, do not use public directories because users can easily overwrite each others' changes.

For detailed information about using group security with DCM, see the *Rational Synergy Distributed* book

Options and arguments

`-a|-assign` *`object_spec`*`:readsource`

Assigns you to assign security for a specified object. Uses the default text editor to add groups. If the `group_name` contains spaces, quotes must be used. If `:readsource` is used, that object has read security (the source file cannot be viewed).

`-c|-create` *`group_name`*

Creates a group name. Uses the default text editor to define the group membership.

`-e|-edit` *`group_name`*

Edits an the user membership of an existing named group.

`groupname`

Specifies the group name, with one or more items separated by spaces, tabs, and/or commas. If `groupname:readsource` is specified, users in the specified named group are granted read access to the source contents.

`-l|-list`

Lists all the defined groups.

`-v|-value` *`groupname`*

Shows the groups available to the specified object.

Example

- List all defined groups.

  ```
  ccm groups -l
  ```

- Create a group named *`docs`*.

  ```
  ccm groups -c docs
  ```

- Show the groups available to an object named `makefile.pc-1:makefile:tut63#4`.

  ```
  ccm groups -value makefile.pc-1:makefile:tut63#4
  qa_team
  design_team
  ```

help command

Synopsis

```
ccm help [argument]
```

Description and uses

Use the `help` command to obtain online help. Note the following ways that you can receive general information about help:

- Without an argument, the `ccm help` command displays a list of Rational Synergy commands.
- Entering `ccm alias` displays Rational Synergy command aliases.

Online help is invoked in the following ways.

- Windows users

  Requesting help from the command line in the form of `ccm help` command causes Rational Synergy to invoke the default browser, and then display help for the command requested.

- UNIX users (running a GUI process; the terminal window from which you issued the command has its `DISPLAY` environment variable set correctly)

  Requesting help from the command line in the form of `ccm help` command causes Rational Synergy to invoke the default browser, and then display help for the command requested.

- UNIX users (not running a GUI process, or running a GUI process but with the `DISPLAY` environment variable not set in the terminal window from which you issued the command)

  Requesting help from the command line in the form of `ccm help` command causes Rational Synergy to display an ASCII text translation of the HTML help file.

Options and arguments

argument

Specifies the name of the command for which you want help.

Example

- Request help on the `type` command.

  ```
  ccm help type
  ```

history command

Synopsis

```
ccm hist|history [-f|-format "format_string"]
                 file_spec [file_spec...]
ccm hist|history [-f|-format "format_string"]
                 -p|-project project_spec [project_spec...]
```

Description and uses

Use the `history` command to show the version history of an object. With the `-g` option, the version history is displayed graphically; otherwise, it is displayed in the format of a log report.

Options and arguments

`-f|-format "format_string"`

Specifies the format of the output. The required string uses keywords and literal text, such as:

```
%displayname %owner
```

A keyword can be built-in (`%fullname`, `%displayname`, `%objectname`) or the name of any existing attribute such as `%modify_time` or `%status`.

See Built-In keywords for a list of keywords.

`file_spec`

Specifies the name of the file or directory for which the history is being shown.

`-g`

Brings up the appropriate dialog. You must specify the *`file_spec`* or *`project_spec`* for the object whose history you want to view, otherwise you will receive an error message.

`-p|-project` *`project_spec`*

Shows the history of a project.

Examples

- Examine the history of `main.c` from within the parent project's work area.

```
ccm history main.c
Object:  main.c-1 (csrc:2)
Owner:   bob
State:   integrate
Created: Tue Jun 4 13:04:23 1999
Task:    4
Comment:
Predecessors:
Successors:

   main.c-2:csrc:2
************************************************************
Object:  main.c-2 (csrc:2)
Owner:   john
State:   integrate
Created: Mon Jun 24 18:02:22 1999
Task:    7
Comment:
Predecessors:
      main.c-1:csrc:2
Successors:
      main.c-3:csrc:2
************************************************************
Object:  main.c-3 (csrc:2)
Owner:   bob
State:   working
Created: Mon Aug 12 18:03:31 1999
Task:    12
Comment:
Predecessors:
      main.c-2:csrc:2
Successors:
************************************************************
```

Related topics

- attribute command
- properties command
- relate command
- unrelate command

import command

Synopsis

```
ccm import -f|-from import_dir_path [-q|-quiet] [-a|-all]
           [-image] [-delimiter delimiter_value]
ccm import {[-f|-from import_dir_path]
           [-n|-name name]
           [-t|-type type]
           [-v|-version version]
           [-i|-instance instance]}
           [-image] [-delimiter delimiter_value]
           [-q|-quiet]
```

Prerequisite

User *ccm_root* must be able to read from the import directory because the Rational Synergy engine process performs the import, and that process runs as user *ccm_root*.

Description and uses

The `import` command imports object versions that are in the Rational Synergy import/export format from the file system into the Rational Synergy database. On Windows, the import directory must be visible to the engine host.

Additionally, you can run this command from a remote client. To do so, use the `-from` option and be sure that the path you specify is visible to the engine host.

When you invoke this command without any arguments or with the `-all` option, all of the exported objects in the directory are imported.

> **Note** This command will overwrite existing objects and tasks in the database without warning.

You must be in the *ccm_admin* role to execute this command.

Options and arguments

`-a|-all`

Imports every object found in the directory.

`-delimiter` *`delimiter_value`*

Specifies the delimiter that separates the parts in an object's four-part name. Use this to import from a database that uses a different delimiter than the one used in the current database. For example, to import from a 4.2.1 UNIX to a 4.4 database, use a colon for *`delimiter_value`*.

The default is "`@`".

`-f|-from` *`import_dir_path`*

Specifies the directory path from which to import. The path you specify must be visible to the engine.

`-image`

Replaces each object's properties in the database with the properties in the import directory: new properties are added, changed properties are replaced, and deleted properties are removed from the database.

`-i|-instance` *`instance`*

Imports every object with an instance of *`instance`* in the directory.

`-n|-name` *`name`*

Imports every object with the name *`name`* in the directory.

`-q|-quiet`

Suppresses some messages output by the command, including warnings about missing relation objects.

`-t|-type` *`type`*`]`

Imports every object of the type *`type`* in the directory.

`-skip_model`

Disables any model handling of imported objects, including any post-handling conversion of objects' data formats to valid formats for this database.

Use this option with care.

`-v|-version` *`version`*`]`

Imports every object with the version *`version`* in the directory.

Examples

Windows:

- Import `foo.c` from the `/users/patty/export_dir` directory.

  ```
  ccm import -from /users/patty/export_dir -n foo.c
  ```

- Import `foo.c` from the `\\fileserver1\export_dir\demo` directory on the `fileserver1` machine. The machine must be visible to the engine process.

  ```
  ccm import -from \\fileserver1\export_dir\demo -n foo.c
  ```

UNIX:

- Import `foo.c` from the `/users/patty/export_dir` directory.

  ```
  ccm import -from /users/patty/export_dir -n foo.c
  ```

Related topics

- export command

license command

Synopsis

```
ccm license [-i|-info] [-t|-timeout minutes]
```

Description and uses

Use the `license` command to display license information, and to set the timeout value.

Options and arguments

`-i|-info`

Displays the current license and the license timeout. This is the default.

`-t|-timeout` *`minutes`*

Sets the license timeout to *`minutes`*.

This option does not apply to sessions with per-user licenses.

Examples

- Set the license timeout to 60 minutes.

  ```
  ccm license -t 60
  ```

- Show whether the current user has a license, or what the license timeout period is.

  ```
  ccm license -i
  ```

lmgr_status command

Synopsis

```
ccm lmgr_status
```

Description and uses

The `ccm lmgr_status` command displays SYNERGY per-user licenses only.

Any user can run this command.

> **Note** The command displays "--" where applicable (no licenses).

Options and arguments

None.

Example

```
ccm lmgr_status
```

Related topics

- message command
- monitor command
- ps command

ln command

Synopsis

```
ccm ln [-s] [-c "comment_string"] [-ce|-commentedit] -cf|-commentfile
       [-t|-task <task_number>]file_path] path_name file_spec
```

Description and uses

The `ln` command operates only on UNIX operating systems.

The `ln` command creates a controlled symbolic link from *`file_spec`* to *`path_name`*.

> **Note** When you create a new link in a non-writable directory, a new directory version is checked out automatically.
>
> If you are in a shared project and your current directory is non-writable, the directory is checked out and associated automatically with the default (or specified) task and is checked in to the *integrate* state. You can disable this feature by setting `shared_project_directory_checkin` to `FALSE` in your initialization file. (See shared_project_directory_checkin.)

Options and arguments

`-c "`*`comment_string`*`"`

Specifies a comment entered in `"`*`comment_string`*`"`.

`-ce|-commentedit`

Brings up an editor for comment entry.

`-cf|-commentfile` *`file_path`*

Specifies where (*`file_path`*) to read the comments for the object.

`file_spec`

Specifies the name of the object from which to create the symbolic link.

`path_name`

Specifies the path to which the symbolic link will point.

`-s`

Provides UNIX-style compatibility; otherwise, the option is ignored.

`-t|-task <task_number>`

Associates the newly created symbolic link with the specified task number.

Example

- Creates symbolic link called `sort.c` to the `sort.c` object in the `ico_2-1` project.

```
ccm ln -s \

/jane/ccm_jane_Aug10/ico_2-1/ico_2/utils/sort.c sort.c
Member sort.c-1 added to project ico_2-1
ccm ln -t 44 /users/kg/ccm_wa/keng421/gdidemo-unix/gdidemo/init.c /users/
gke
Member init.c added to project gdidemo-unix
Associated object version with task 44: init.c-1:symlink:1
Associated object version with task 44: subdirectory-2:dir:1
```

Related topics

- work_area command

ls command

Synopsis

```
ccm ls [-l] [-m] [-R] [-f|-format "format_string"] [file_spec...]
```

Description and uses

The `ln` command operates only on UNIX operating systems.

The `ls` command lists the contents of a directory object version in a work area. By default, the output consists of a list of objects and their associated projections in the file system.

The `ls` command displays two categories of files: objects under Rational Synergy control and files that exist in the file system only. To find out how to display these files, see the `-l` option and the `-m` option.

Options and arguments

`-f|-format "format_string"`

Specifies the format of the output. The format only applies to controlled files. The required string uses keywords and literal text, such as:

```
%displayname %owner
```

A keyword can be built-in (`%fullname`, `%displayname`, `%objectname`) or the name of any existing attribute such as `%modify_time` or `%status`.

See Built-In keywords for a list of keywords.

When `%path` is specified in the format string, all objects are displayed with an absolute work area path. If the work area is not visible, the path is computed.

file_spec

Specifies the file(s) to be displayed.

`-l`

Lists the information in the long format, which contains the status, owner, last modification time, type, instance, name, and version.

If the object is more than six months old, the year is shown instead of the time.

This option causes the following marks to display before the output, when appropriate:

- LC (local copy)

  Denotes files that are in the project, but have a local copy rather than a symbolic link in the work area.

  If files are displayed with this mark and your work area is link-based, you should perform a reconcile operation. For information on the reconcile feature, see the reconcile command.

- NS (not sync'd)

  Denotes files that are in the project, but not in the work area. This situation occurs when you add files to the project, but your work area is not visible, or when a file's link or local copy is deleted.

  If most of the files in your work area are displayed with this mark, you should perform a reconcile operation. For information on the reconcile feature, see the reconcile command.

- UC (uncontrolled)

  Denotes files that are in the work area, but not in the project. To view uncontrolled files marked with UC, you must use the `-m` option with the `-l` option.

To find out advanced information about work areas, read Work area.

`-m`

Shows files that are controlled and uncontrolled. Uncontrolled files are those that are in the work area, but not in the project. To view uncontrolled, marked files, you must use the `-m` option with the `-l` option.

`-R`

Displays subdirectory members recursively. The command does not recurse into subprojects.

Example

List the current directory in the long format.

```
ccm ls -l
working linda Nov 18 11:56 csrc 1 alias.c-4.5
working linda Nov 18 11:56 csrc 1 diff.c-4.5
working linda Nov 18 11:56 csrc 1 move.c-4.5
working linda Nov 18 11:57 csrc  1 start.c-4.5
```

merge command

Synopsis

```
ccm merge [[-create_task] | [-t|-task task_number]]
          [-c|-comment "string"]
          [-ce|-commentedit] [-cf|-commentfile file_path]
          file_spec1 file_spec2
```

Description and uses

When you use the `merge` command to merge source files or directories, the merge tool compares the versions you selected, then compares each version's differences to the closest common ancestor. Rational Synergy creates a new, merged, controlled version automatically when you exit your merge tool.

The merge operation works with both static and modifiable, non-static objects. This feature allows users to merge parallel versions, even when parallel check-in is prevented, in order to use the merge tool. In previous releases, the parallel check-in would not be allowed, so the merge tool could not be invoked.

If you request Rational Synergy to create a new task automatically at the merge, Rational Synergy obtains the task's `release` value from the project in which *`file_spec1`* resides.

File Merge

Each type of object for which you can merge source has default merge tools predefined by Rational Synergy for both the CLI and the GUI.

The automerge tool creates a new, controlled version of your file. If the merge is successful, the merge results are written to the new file.

An area "in conflict" occurs when both versions have changes in the same place relative to the predecessor. If your merged file contains any conflicts, a warning is issued, the tool marks the conflicts so you can find them quickly and easily, then automerge writes the merge results to the new file.

The following example shows how the temporary file is marked:

```
<<<<<<<file1 (file1 changes recommended)
unique lines in file1
======= (common lines)
unique lines in file2
>>>>>>>file2 (file2 changes recommended)
```

Directory Merge

From the command line, the merge tool automatically includes all members from both directories in a new, controlled, merged directory. If one of the objects to be merged is a

member of the current project, Rational Synergy uses the new merged directory in the project. This applies to both root directories and subdirectories.

Options and arguments

`-c|-comment "`*`string`*`"`

Specifies the comment string.

`-ce|-commentedit`

Brings up the default editor.

`-cf|-commentfile` *`file_path`*

Uses the comments from the specified file. If you specify both a comment string and comment file, the comments are merged; the comment string will follow the comments from the file.

`-create_task`

Causes a task to be created automatically when Rational Synergy creates the new, merged object version, and associates the new object version with that task.

The task is assigned to the user who performed the merge. The task's release value is set to the release value of the project in which the new object version is created. If the object version is created outside of a project, the release value is not set.

`file_spec1`

Specifies the name of the first file or directory to be merged.

`file_spec2`

Specifies the name of the second file or directory to be merged.

To bring up the Merge dialog, you must specify the two files and their versions to be merged, otherwise you will receive an error message. If the objects to be merged are directories, the Directory Merge dialog is displayed.

`-task` *`task_number`*

Associates the newly created, merged object with the specified task.

If you have set the current (default) task and do not specify a different task in this field, the merged object is associated with the current task automatically.

Example

- Merge two files: `bufcolor.c-4` and `bufcolor.c-2.1.2`.

```
ccm merge bufcolor.c-4 bufcolor.c-2.1.2
The current task is set to:
ccmint24#22: Merge 'bufcolor.c-4' (associated with task '21') ...
Task 'ccmint24#22: Merge 'bufcolor.c-4' (associated with task '21') ...'
was created with a release value of 1.1.
Merging attribute source between objects:
        bufcolor.c-4 and bufcolor.c-2.1.2
        with ancestor bufcolor.c-2.1.1.
        Merge Source in progress...
Parallel versions exist for bufcolor.c-4:csrc:ccmint24#1
Adding 'release' attribute with value '' to object bufcolor.c
5:csrc:ccmint24#1
Warning: Merge Source warning. (overlaps during merge).
        Merge conflicts have been noted in bufcolor.c-5.
        Search for '<<<<<<<' to find conflicts.
Merge Source completed successfully.
        Merged object is bufcolor.c-5
Associated object version with task 22: bufcolor.c-5:csrc:ccmint24#1
Using file 'bufcolor.c-5' in project 'test-laura'...
Replaced 'bufcolor.c-2.1.2' with 'bufcolor.c-5' in project 'test-laura'...
```

- Merge two files and create a new task.

  When you merge two object versions, Rational Synergy takes the data in each file, then creates a third object version. The third object gets its version from the first object version you specify from the command line.

  `ccm merge` *`file_spec1 file_spec2`* `-create_task -comment "`*`Your comment.`*`"`

  If you do not know which version to merge with, use the following `query` command to find the correct object version.

  `ccm query -o` *`username`* `-v` *`version`* `-s` *`state`* `-t` *`type`*

- Merge two directories.

  `ccm merge` *`directory1-version directory2-version`*

Related topics

- diff command

message command

Synopsis

```
ccm message -u|-user username "message_text"
            -d database_path "message_text"
            -rfc_address address "message_text"
            -attr value "message_text"
```

Description and uses

The `ccm message` command communicates a message directly or by broadcast to users who are running a Rational Synergy session.

Messages are prepended with the name of the user who sends the message. If you want to send messages to specific sessions, use the `/rfc_address` switch.

Options and arguments

-attr value

Provides an additional criterion for the message destination. The possible *attr* option is replaced with one of the following values: `process, callback, display, pid, user, host, database, engine_address`, and `pwa_path`.

You can use only one switch at a time.

-d *database_path*

Specifies the database to which the message is sent.

You can use an ACCENT regular expression to specify multiple databases. The ACCENT regular expression must contain a leading question mark (?) character.

-rfc_address *address*

Specifies the RFC (remote function call) to which the message will be sent. The format starts with a host name or IP address, such as `host:port[:ip]` or `ip:port[:ip]` `[:ip]` represents zero or more IP addresses.

> **Note** If the specified `rfc_address` is for an engine, the `ccm message` command will fail.

-user *user*

Specifies the user to whom the message will be sent.

Example

- Execute the `ccm message` command using the `host` attribute:

  ```
  ccm message -host comp1 "New compile server is up"
  ```

- Use the `database` attribute to specify a message to all databases:

  ```
  ccm message -d "?" "Server going down in 2 minutes..."
  ```

Caveats

If the `-rfc_address` specified is for an engine, the command will fail.

Related topics

- fs_check command
- monitor command
- ps command
- diff command

migrate command

Synopsis

Migrate

```
ccm migrate -d|-dir|-directory dirname -p|-project project_spec
            [-a|-arch_state arch_state]
            [-cb|-copy_based|-not_copy_based|-ncb] - UNIX only
            [-dt|-default_type type]
            [-i|-import_identical]
            [-mc|-meta_create_time]
            [-mo|-meta_owner]
            [-mr|-meta_release]
            [-ms|-meta_status]
            [-n|-nomerge]
            [-nt|-notask]
            [-path|-set|-setpath]absolute_path
            [-r|-release release_value]
            [-rel|-relative|-nrel|-not_relative]
            [-rl|-rules filename]
            [-st|-state state]
            [-t|-task task_number]
            [-tl|-translate|-ntl|-no_translate]
            [-wa|-maintain_wa|-nwa|-no_wa]
            [-wat|-wa_time |-nwat|-no_wa_time]
```

Description and uses

The `migrate` command loads a directory structure from a file system into Rational Synergy.

Any user can execute this command.

For additional information about the migrate process, see Migration rules.

Options and arguments

`-a|-arch_state arch_state`

Specifies the state (status) that the PVCS® (Windows) or RCS and SCCS (UNIX) archived object versions will have after you load them into Rational Synergy.

You can set this value in the initialization file for all migrates. See migrate_default_arch_state for more information.

`-cb|-copy_based`

Makes the work area copy-based (UNIX only) .

This is a work area option. See work_area command for more information.

`-d|-dir|-directory` *`dirname`*

Specifies the root directory to be migrated.

`-dt|-default_type` *`type`*

Specifies the object type for files that have no migration rules. For example, if `default_type` is set to `binary` and there are no migration rules for `*.pdf` files, `*.pdf` files are migrated as `binary` files.

Incremental migration of a new version of an existing object will keep the same type.

You can set this value in the initialization file for all migrates. See migrate_default_type for more information.

`-g`

Brings up the appropriate dialog.

`-i|-import_identical`

Causes new versions to be created for files in the file system that are identical to their corresponding objects in the database. (The file and the latest version of its corresponding object are "identical" if their source codes are identical.) Usually, you skip identical versions (the default); however, you might choose to load identical versions if you are migrating a new release of another vendor's software.

By default, identical files are skipped.

`-mc|-meta_create_time`

Causes migrated files' Rational Synergy create time attributes to be set to their third-party archive create times: PVCS (Windows), or SCCS or RCS9 (UNIX).

Note You must be in the *ccm_admin* role to use this option.

`-mo|-meta_owner`

Causes migrated files' Rational Synergy owner attributes to be set to their third party archive owner: PVCS (Windows), or SCCS or RCS9 (UNIX).

Note You must be in the *ccm_admin* role to use this option.

`-mr|-meta_release`

Causes migrated files' Rational Synergy release attributes to be set to their third-party archive release values: PVCS (Windows), or SCCS (UNIX).

Rational Synergy does not ensure that the third-party archiver's release value is a valid release before setting a migrated file's release value. You must add the release manually if the value does not exist.

Caution for Windows - If a PVCS file has more than one version (release value) label, the labels are concatenated into one space-delimited line that you must edit after migrating the file.

`-ms|-meta_status`

Causes migrated file's Rational Synergy state attributes to be set to their third-party archive state values: PVCS (Windows), or SCCS or RCS9 (UNIX).

Note You must be in the *ccm_admin* role to use this option.

Caution You must define the state as a SYNERGY status.

`-n|-nomerge`

Causes new versions of directory objects to replace old versions. The result is that the new directory object's list of members contains only the members just migrated instead of a merged list of the old and new members.

`-ncb|-not_copy_based - UNIX only`

Makes the work area link-based.

This is a work area option. See work_area command for more information.

`-nrel|-not_relative`

Windows: Makes subprojects' work areas absolute instead of relative to the parent project's work area. This is the default when you first create a project.

UNIX: Causes links to be used for subprojects and makes subprojects' work areas absolute instead of relative to the parent project's work area. This is the default when you first create a project.

This is a work area option. See work_area command for more information.

`-nt|-notask`

Causes the migrate operation to ignore task requirements.

By default, the migrate operation enforces task requirements for creating and checking out new versions, and for checking in checked-out versions' predecessors. The task requirement is defined in each type's Require Task At option.

`-ntl|-no_translate`

If you are using a UNIX server and a Windows client, preserves the Windows format of `ascii`-type object sources. For example, Windows newlines in a `csrc` object are preserved instead of being converted to UNIX newlines.

This is a work area option. See work_area command for more information.

`-nwa|-no_wa`

Causes the work area not to be maintained (that is, disconnects the migrated project's work area from the database).

This is a work area option. See work_area command for more information.

`-nwat|-no_wa_time`

Use this option only if you are going to stop performing third-party builds in this project, or you previously have set Use New Timestamps in the GUI, or used the `-wa_time` option on the command line. See the work_area command for more work area information.

Ensures that the objects' timestamps are set to their Rational Synergy modification (database creation or use) times instead of their file system times.

`-p|-project` *`project_spec`*

Specifies the project and version into which the objects will be migrated. Use a unique name if you are creating a new project. If you are performing an incremental migrate, make sure you use the existing project name and hierarchy.

Note If the name you use matches an existing name, your project will be overwritten with the migrated objects. No warning is given before the objects are changed.

`-path|-set|-setpath` *`absolute_path`*

Changes the specified project's work area path to the new location.

This is a work area option. See work_area command for more information.

`-r|-release` *`release_value`*

Specifies a release value for the migrated files and directories. Possible candidates include components and component release values used in the database, plus other release values used in your database.

`-rl|-rules` *`filename`*

Specifies the name of the optional rules file to use. You can set up an unlimited number of rules files, but you can use only one at a time.

See Migration rules for more information about the rules.

`-rel|-relative`

Makes the work area path relative to the parent project's path. You can set this option if the project is used in only one place. After you set the project's work area path to relative, you cannot use the project in more than one project.

Directories migrated as subprojects are affected by this option because they have parent projects. Parent projects are not affected.

This is a work area option. See work_area command for more information.

`-st|-state` *`state`*

Specifies the state (status) given to the migrated files.

If you set the archive state to be a writable state, the history of the migrated object will not be preserved if the object is an archived object.

When you migrate an SCCS or RCS file using a non-writable initial state, all the file's versions are migrated into the database. The history of the resulting file object shows a preserved version history; that is, successor versions are checked out from previous versions, resulting in the preservation of the archive's history. If you choose a writable state for the import state of archive types, an object version is created for the specified file instead of being checked out from the previous version because you cannot check out from a writable version. The result is that you cannot preserve the archive's history unless the archive import is non-writable.

You can set this value in the initialization file for all migrates. See migrate_default_state for more information.

`-t|-task` *`task_number`*

Specifies a task with which to associate newly checked-out or created object versions.

`-tl|-translate`

If you are using a UNIX server and a Windows client, converts `ascii`-type object source from the Windows format to the UNIX format. For example, Windows newlines in a `csrc` object are converted to UNIX newlines.

This is a work area option. See work_area command for more information.

`-wa|-maintain_wa`

Causes the work area to be maintained (that is, synchronizes the work area and keeps it synchronized).

This is a work area option. See work_area command for more information.

`-wat|-wa_time`

Ensures that the objects' timestamps are set to their file system times instead of their Rational Synergy modification (database creation or use) times.

Use this option only if you are going to perform third-party builds in this project. See the work_area command for more work area information.

Examples

Windows:
Migrate a project named `ico-1` from the directory `C:\examples\ico\man`, with the migration rules located in `C:\rules\migrate.rul`.

```
ccm migrate -p ico-1 -d C:\examples\ico\man -rl C:\rules\migrate.rul
```

UNIX:
Migrate a project named `ico-1` from the directory `/usr/local/ccm/examples/ico/man`, with the migration rules located in ~/migrate.rul.

```
ccm migrate -p ico-1 -d /usr/local/ccm/examples/ico/man -rl ~/migrate.rul
```

- Migrate the *directory_path* directory hierarchy into the *project_name-version* project with the object and archive states set to *state* and *archive_state*, respectively.

  ```
  ccm migrate -p project_name-version -d directory_path -st state -a
  arch_state
  ```

- Migrate the *directory_path* directory hierarchy into the *project_name-version* project, with the default object type set to `html`.

  ```
  ccm migrate -p project_name-version -d directory_path -dt html
  ```

- Migrate the *directory_path* directory hierarchy into the *project_name-version* project, and load identical versions.

  ```
  ccm migrate -p project_name-version -d directory_path -i
  ```

monitor command

Synopsis

```
ccm monitor -u|-user username
            -d|-database database
            -rfc_address address
            -attr value
```

Description and uses

The `ccm monitor` command provides a network-wide view of Rational Synergy user and process information including:

- user
- process type (engine, user interface, router, license manager, or object registrar)
- host
- port
- process ID
- database path

The `ccm monitor` command appends an exclamation point (!) to the status field of a process when that process has not responded to the router for a fixed amount of time. This failure to respond is assumed to indicate a problem (for example, the machine that was running the process has gone down or the process is hung).

If a delay in response is due to a busy machine, the exclamation point will go away when the operation that is using the machine is finished.

The monitor view may refresh, depending on what is changing.

To stop monitoring on UNIX, press `CTRL+C`.

Options and arguments

`-attr value`

Specifies the name of the field to be monitored. The possible *`attr`* option is replaced with one of the following values: `process`, `display`, `pid`, `user`, `host`, `database`, `engine_address`, and `pwa_path`.

You can use only one switch at a time.

`-database` *`database`*

Specifies that all users of *`database`* are monitored.

You can use a regular expression to monitor multiple databases. The regular expression must contain a leading question mark (?) character.

`-rfc_address` *address*

Specifies the remote function call (RFC) address of the Rational Synergy interface (GUI) process to be monitored. The format starts with a host name or IP address, such as `host:port[:ip]` or `ip:port[:ip]`. `[:ip]` represents zero or more IP addresses.

> **Note** If the specified `rfc_address` is for an engine, the `ccm monitor` command will fail.

`-user` *user*

Specifies the user to be monitored.

Example

Monitor user *kim*'s engine processes.

```
ccm monitor -user kim -process engine

Rational process monitor...7 process(es) located:
user     process  host     port   pid    database path
----     -------  ----     ----   ---    -------------
ccm_root router   galaxy   1514   28496  -
ccm_root objreg   orbit    34525  18273  -
ccm_root objreg   galaxy   41587  28507  -
ccm_root objreg   dbserver 62240  19592  -
linda   engine   lego    34728  21182 /vol/dbserver/ccmdb/ccm51new
linda   gui      lego    34725  21181 /vol/dbserver/ccmdb/ccm51new
linda   monitor  lego    34737  21205  -
[Press ^C to quit Rational monitor.]
```

Related topics

- fs_check command
- lmgr_status command
- message command
- ps command

move command

Synopsis

```
ccm move file_spec [file_spec...] dest_directory
         [-task task_number]
ccm move directory [directory...] dest_directory
         [-task task_number]
ccm move file_spec new_file_spec [-task task_number]
ccm move -p|-project project_spec new_project_spec
```

Description and uses

The `move` command has the following uses:

- Renames a file or project. Once you rename the project, the root directory is renamed automatically to reflect the project's new name.

  Note If you attempt to rename a project that is writable by you, but that has a root directory that is non-writable, the operation will fail. You must first check out the root directory. By doing so, when you rename the project, Rational Synergy will rename the root directory automatically.

- Moves one or more files to another directory.
- Moves a file to a new project (in a new work area).
- Moves one or more directories and their contents to a particular directory.
- Moves a subproject to a new top-level project.
- Moves one or more subprojects and their contents to another directory.

  Note When you move an object to or from a non-writable directory, one of the following occurs:

  Rational Synergy checks out a new directory version automatically. You must check **in** the directory to make it available to other users. If you are using task-based methodology, this is done automatically when you check in the default (or specified) task. If you are using object-status-based methodology, you **must** remember to check in the directory.

  If you are in a shared project and your current directory is non-writable, Rational Synergy checks out the directory and associates it automatically with the default (or specified) task, then checks it in to the *integrate* state. You can disable this feature by setting

`shared_project_directory_checkin` to `FALSE` in your initialization file. (See shared_project_directory_checkin.)

You do not need to be in a work area to use this command as long as you use the Project reference form.

Note If you attempt to rename a project or object used in other directories or directory versions besides the current or specified directory, you are reminded to check out a new version of the object. The rename of an object requires the modification of all directories binding the specified object, but only the current or specified directory can be safely updated.

Options and arguments

dest_directory

Specifies the name of the directory to which the file or directory is being moved.

directory

Specifies the name of the directory to be moved.

file_spec

Specifies the name of the file or directory that is being moved.

-p|-project *project_spec new_project_spec*

Specifies the name of the project that you want to rename and the new name.

You **must** use this option if you are renaming a top-level project.

You cannot use this option to rename a project that is in use as a subproject.

-task *task_number*

Associates any newly created directory with the specified task.

If the current (default) task is set and you do not specify a different task, the newly created directory is associated with the current task automatically.

Examples

- Move the file `oops.h` from the `src` directory to the `incl` directory in your current project.

  Windows:
  ```
  ccm move src\oops.h incl/
  ```
  UNIX:
  ```
  ccm move src/oops.h incl/

  Member oops.h-3 removed from the project sandbox-lb
  Adding 'release' attribute with value '2.0' to object incl-2:dir:5
  Associated object incl-2:dir:5 with task 36.
  Member oops.h-3 added to project sandbox-lb
  ```

- Rename the file `turquoise.c` to `magenta.c` in the current project.

  ```
  ccm move turquoise.c magenta.c
  ```

- Rename the `ccm_aug8-1` project to `test_a-1`.

  Windows:
  ```
  ccm move -p ccm_aug8-1 test_a-1
  Setting path for work area of 'test_a-1' to
  'c:\users\linda\ccm_wa\ccmint07\test_a-1'...
  ```
  UNIX:
  ```
  ccm move -p ccm_aug8-1 test_a-1
  Setting path for work area of 'test_a-1' to '/linda/ccm_wa/ccmint07/
  test_a-1'...
  ```

- Rename the `hello.c` file to `hi_world.c`, then move it to another project's directory.

  Windows:
  ```
  ccm move proj\hello.c@proj-1 screen\src\hi_dir\hi_world.c@beta-3
  ```
  UNIX:
  ```
  ccm move proj/hello.c@proj-1 screen/src/hi_dir/hi_world.c@beta-3

  Member hi_world.c-1 removed from project proj-1
  Member hi_world.c-1 added to project beta-3
  ```

- Move the file `hello.c` from `beta-1` to a new project called `final-1`.

  Windows:
  ```
  ccm move beta-1\hello.c@beta-1 final@final-1
  ```
  UNIX:
  ```
  ccm move beta-1/hello.c@beta-1 final@final-1

  Member hello.c-1 removed from project beta-1
  Member hello.c-1 added to project final-1
  ```

ps command

Synopsis

```
ccm ps -user username "message_text"
        -d|-database database "message_text"
        -rfc_address address "message_text"
        -attr value "message_text"
```

Description and uses

The `ccm ps` command provides network-wide process status information on Rational Synergy users and processes. It is a maximum verbosity version of `ccm monitor`.

Options and arguments

`-attr value`

Specifies the name of the field to be monitored. The possible *attr* option is replaced with one of the following values: `process, display, pid, user, host, database, engine_address, and pwa_path.`

You can use only one switch at a time.

`-d|database database`

Specifies that all uses of *database* are monitored.

You can use a regular expression to monitor multiple databases. The regular expression must contain a leading question mark (?) character.

`-rfc_address address`

Specifies the remote function call (RFC) address of the Rational Synergy interface (GUI) process to be displayed. The format starts with a host name or IP address, such as `host:port[:ip]` or `ip:port[:ip] [:ip]` represents zero or more IP addresses.

> **Note** If the specified `rfc_address` is for an engine, the `ccm process` command will fail.

`-user username`

Specifies the user to be monitored.

Example

- Display the process information for the interface with a host address of "horse.abbd0.com".

  ```
  ccm ps -host horse.abcc0.com
  ```

- Display the process information for databases that have names containing "training".

  ```
  ccm ps -d "?training"
  ```

Related topics

- fs_check command
- message command
- monitor command

process_rule command

Synopses

Add Folders and/or Folder Templates to a Process Rule

```
ccm pr|process_rule|ut|update_temp|update_template|
           rt|recon_temp|reconfigure_template -a|-ad|-add
       [-fol|-folder|-folders folder_specs]
       [-ft|-folder_temp|-folder_temps|
       -folder_template|-folder_templates folder_template_specs]
       [-q|-quiet] process_rule_specs
```

Copy a Process Rule

```
ccm pr|process_rule|ut|update_temp|update_template|
           rt|recon_temp|reconfigure_template -cp|-copy
      process_rule_spec1 process_rule_spec2
```

Compare Two Process Rules

```
ccm pr|process_rule|ut|update_temp|update_template|
            rt|recon_temp|reconfigure_template
      -comp|-compare process_rule_spec1 process_rule_spec2
```

Delete Process Rules

```
ccm pr|process_rule|ut|update_temp|update_template|
             rt|recon_temp|reconfigure_template -d|-delete [-force]
      process_rule_specs
```

List Process Rules

```
ccm ut|update_temp|update_template|
            rt|recon_temp|reconfigure_template -l|-list
```

Modify a Process Rule

```
ccm pr|process_rule|ut|update_temp|update_template|
              rt|recon_temp|reconfigure_template -m|-modify
       [-fol|-folder|-folders folder_specs]
       [-ft|-folder_temp|-folder_temps|-folder_template|
       -folder_templates folder_template_specs]
       [(-bn|-baseline_name baseline_name)|
       (-lb|-latest_baseline)|(-usb|-user_selected_baseline)|
       (-lbp|-latest_baseline_projects)]
       [-brp|-baseline_release_purpose|
       -baseline_release_purposes release_purposes [(-pr|-prepend)|
```

```
        (-ap|-append)]
        [(-pb|-prep_baseline)| (-nopb|-noprep_baseline)]
        [-matching "version_matching_string"]
        [-default|-nodefault]
              process_rule_specs
```

Remove Folders and/or Folder Templates from a Process Rule

```
ccm pr|process_rule|ut|update_temp|update_template|
              rt|recon_temp|reconfigure_template -rem|-remove
       [-fol|-folder|-folders folder_specs]
       [-ft|-folder_temp|-folder_temps|-folder_template|
       -folder_templates folder_template_specs]
       [-q|-quiet] process_rule_specs
```

Set the Controlling Database for a Process Rule

```
ccm pr|process_rule|ut|update_temp|update_template|
             rt|recon_temp|reconfigure_template -cdb|-controlling_database
       [-local|-handover dbid|-accept dbid]
       process_rule_specs
```

Show Information about Process Rules

```
ccm pr|process_rule|ut|update_temp|update_template|
             rt|recon_temp|reconfigure_template -sh|-show keyword
      process_rule_specs
```

Description and uses

The `process_rule` command displays and sets process rules. Note that the `process_rule` command was referred to as the `update_template` and `reconfigure_template` commands in prior releases of Rational Synergy.

A process rule specifies how a project will be updated when an update operation is performed on the project. The combination of a project's purpose and release value determines which process rule can be used in the project. Multiple process rules can be created for a release/purpose pair. This allows you to set up rules, and then select rules to apply to a given release and purpose, and to switch among the set of process rules during the course of a release. It also allows you to reuse process rules for future releases, rather than have to continually modify the process rule for each purpose.

Process rules are automatically created whenever one of the following occurs.

- When a build manager creates a new release value, a process rule is created for each valid purpose that is associated with that release value.
- When a build manager adds a new purpose to the list of valid purposes for a particular release value, a process rule is created for that unique combination of project purpose and release value.

- When a build manager creates a new purpose, a general process rule is created for that purpose. The build manager must then edit this new (empty) process rule.
- When a build manager copies a process rule.

General process rules are shipped with the product for both standard and distributed processes. A non-DCM-initialized database contains the standard process, and a DCM-initialized database contains both the standard and the distributed processes. The process rules have the same behavior in each, with the following exceptions:

- In the standard process, Collaborative Development collects all completed tasks from all databases, while in the distributed process, Local Collaborative Development collects all completed tasks from the local database.
- In the standard process, Integration Testing collects all completed tasks from all databases, while in the distributed process, Local Integration Testing collects all completed tasks from the local database, and master integration tested tasks from foreign databases.

The standard process is used to provide process rules in synergy classic and the CLI when a purpose is specified.

You can specify which process rule is to be used when you create a new release. For more information, see the release command.

Use this command to perform the following operations.

- Edit a process rule.
- Add folders and/or folder templates to a process rule.
- Remove folders and/or folder templates from a process rule.
- Copy a process rule.
- Compare two process rules.
- Delete one or more process rules.
- List the currently-defined process rules.
- Set the controlling database for process rules.
- Show information about one or more process rules.

Options and arguments

`-accept` *`dbid`*

Specifies that the object is set to accept control from the specified database. This enables you to set up central administration of process rules using DCM.

-bn|-baseline_name *baseline_name*

Specifies the name of the baseline (*baseline_name*) you want to use for your process rule.

-brp|-baseline_release_purpose|-baseline_release_purposes *release_purposes* [(-pr|-prepend) | (-a|-append)]

Specifies the list of release-purpose pairs for the process rule. The *release_purposes* value is a list of one or more release-purpose items that are separated by commas. A release-purpose item consists of a release value, a colon (:), and a purpose value. The release value may be a valid release name or the keywords %release or %baseline_release. The purpose value may be a valid purpose as defined in the project purpose table, or the value Any. The release-purpose items are specified in decreasing order of search priority. If the -pr|-prepend option is specified, the specified values are inserted at the beginning of the existing list. If the -a|-append option is specified, the specified values are appended at the end of the existing list. If neither option is specified, the specified values replace the current list.

-cdb|-controlling_database

Specifies that the specified process rule is to have a controlling database. Use with the -local, -handover, or -accept option to specify where the template is defined. Use these options to set up centralized administration for process rules.

-comp|-compare *process_rule_spec1 process_rule_spec2*

Compares two process rules.

-cp|-copy *process_rule_spec1 process_rule_spec2*

Copies a specified process rule to a new process rule or copies over an existing one. The following rules apply when copying process rules:

Generic to generic copies:
If a generic process rule is copied to an existing generic process rule, the target process rule keeps the old name (the four-part name and the case_preserved_name attribute), but all other properties are copied from the source process rule.

You can copy a generic process rule to a new generic process rule.

Generic to release-specific copies:
If a generic process rule is copied to an existing release-specific process rule, the target process rule keeps the old name (the four-part name, the

`case_preserved_name` attribute, and the `release` attribute) and its old association to a generic process rule. All other properties are copied from the source process rule.

Release-specific to release-specific copies:
If a release-specific process rule is copied to an existing release-specific process rule, the target process rule keeps its old name (the four-part name, the `case_preserved_name` attribute, and the `release` attribute), but all other properties are copied from the source process rule.

The target release-specific process rule also keeps its existing association with a generic process rule.

Release specific to generic copies:
You cannot copy a release-specific process rule to a generic process rule.

You must be working as an Process rules manager to use this option.

`-default|-nodefault`

If `-default` is specified, causes all new projects that have the same release value and purpose combination as the process rule you are editing to automatically use the process rule. If `-nodefault` is specified, all new projects that have the same release value and purpose as the process rule will not use the process rule and will update manually.

If you do not specify this option, the default is `-default`.

`-d|-delete` *`process_rule_specs`*

Deletes the specified *`process_rule_spec`*. If the specified process rule is generic or is used by one or more projects, it is deleted only if you specify the `-force` option.

`-fol|-folder|-folders` *`folder_spec`*

Specifies the IDs of the folders that you are adding or removing. For this argument's syntax, see Folder specification.

> **Note** You can specify the name of a file that contains a *`folder_spec`* wherever you can specify *`folder_spec`*.

`-force`

The `-force` option suppresses confirmation messages and forces the delete operation to be carried out.

`-ft|-folder_temp|-folder_temps|-folder_template|-folder_templates` *`folder_template_specs`*

Specifies the folder templates that you are adding or removing.

`-handover` *`dbid`*

Specifies that control of the object is handed over from the current database to the specified database. Use this option to set up centralized administration for process rules using DCM.

`-lb|-latest_baseline`

The baseline with the latest creation time, and with matching *`baseline_release`* and *`baseline_purpose`*, is used as the baseline for instantiations of the template. If this option is used, the following options cannot be used: `-baseline_name`, `-latest_baseline_projects`, `-prep_baseline`, and `-matching`.

When this option is used, the currently-selected process rule chooses the baseline from a non-empty pool of baselines that match one of the release/purpose pairs. When a baseline project cannot be selected using `latest_baseline`, the project will not have a baseline.

`-lbp|-latest_baseline_projects`

Each project that uses this template will select its own individual project baseline at the time that the update operation is carried out. The baseline project is the one with matching *`baseline_release`* and *`baseline_purpose`*, and with a version that matches *`version_matching_string`*. The `-prep_baseline` option and the `-matching` option may be used with this option only.

`-local`

Specifies that local control is set, which breaks any previous DCM replication from another database.

`-l|-list [`*`scope`*`]`

Lists the currently-defined process rules.

`-matching "`*`version_matching_string`*`"`

Matches the appropriate baselines to projects that use process rules.

Enables you to enter a version that can be used to identify the baseline. Use this field if specifying the release of the baseline is insufficient because you have more than one release version of a project with the same release value.

For example, let's say a company has three released project hierarchies all for release 1.0: the project versions are `1.0_alpha`, `1.0_beta`, and `1.0_gr`. In this case, specifying the Baseline Release option as 1.0 is not enough to identify projects that use this process rule. You would want to set the Baseline Versions Matching option to `1.0_gr` to indicate that the project with a version of `1.0_gr` should be used as the baseline.

If all baselines in the `1.0_gr` project hierarchy do not have identical versions, but their versions are similar, you can specify a wildcard. For example, if your project hierarchy contains projects with versions `1.0_gr`, `1.0_gr_unix`, and `1.0_gr_windows`, you could set the Baseline Versions Matching option to `1.0_gr*`. This setting would select the version with the prefix `1.0_gr`, even though the remainder of the version might differ. (If a project has more than one choice for a baseline, it will select the baseline whose platform matches. For example, project `2.0_int_unix` might identify `1.0_gr_unix` and `1.0_gr_windows` as potential baselines, but it will check for a matching platform, then use `1.0_gr_unix`. This is because Rational Synergy is set up to support development of parallel platforms by default.)

`-modify`

Modify the properties of an existing process rule.

`-nopb|-noprep_baseline`

This option is only relevant when the process rule has a baseline selection mode of `latest_baseline_projects`. It indicates that *prep* state projects are not to be considered as potential baseline projects for the individual projects that use this process rule.

`-pb|-prep_baseline`

This option is relevant only when the process rule has a baseline selection mode of `latest_baseline_projects`. It indicates that static projects and *prep* state projects are to be considered as potential baseline projects for the individual projects that use this process rule.

`-quiet`

Reduces the number of output messages that are displayed. This option is useful for scripting.

`-rem|-remove`

Removes the specified task or folder from the process rule.

`-sh|-show` *`keyword`*

Shows information about the specified process rule.

The following keywords are supported by this option:

- `baseline_projects`

  Shows the baseline projects that match the process rule's baseline properties.
- `brp|baseline_release_purpose|baseline_release_purposes`

  Shows the baseline release and purposes for the process rule.
- `bsm|baseline_selection_mode`

  Shows the baseline selection mode, which is one of the following:

  `Baseline_name`

  `Latest Baseline`

  `User-Selected Baseline`

  `Latest Baseline Projects`
- `default`

  Shows whether the process rule is used for new projects by default.
- `fol|folder|folders`

  Shows the folders that are used by the process rule.
- `ft|folder_temp|folder_temps|folder_template|folder_templates`

  Shows the folder templates that are used by the process rule.
- `i|info|information`

  Shows all properties of the specified template.
- `matching`

  Shows the version-matching string for the process rule.
- `members`

  Shows the tasks, folders, and folder templates that are used by the process rule.
- `pb|prep_baseline`

  Shows whether a *prep* baseline is an eligible baseline for the process rule.

`-usb|-user_selected_baseline`

Specifies that the process rule does not specify a baseline that is to be used to find baseline projects. The baseline is selected by the user.

Related topics

- process_rule examples

process_rule examples

View examples for the following operations:

- Add Folder and/or Folder Templates
- Compare Two Process Rules
- Copy a Process Rule
- Delete Process Rules
- Modify a Process Rule
- Set the Controlling Database for Process Rules
- Show Information about Process Rules

Add Folder and/or Folder Templates

- Add folder 3 to the `2.1:Insulated Development` process rule:

```
ccm pr -add -folders 3 "2.1:Insulated Development"

   Added the following folder(s) to process rule 2.1:Insulated
Development
      Folder 3
   Added 1 folder to process rule 2.1:Insulated Development.
```

- Add folder template `xxx` to the `2.1:Insulated Development` process rule.

```
   ccm pr -add -folder_temp "integration tested tasks for release
%release" "2.1:Insulated Development"

   Added the following folder template(s) to process rule 2.1:Insulated
Development
      Folder template integration tested tasks for release %release
   Added 1 folder template to process rule 2.1:Insulated Development.
```

Compare Two Process Rules

- Compare the process rules for the `toolkit/2.0:Collaborative Development` project and the `toolkit/2.0:Insulated Development` project.

```
   ccm process_rule -compare "toolkit/2.0:Collaborative Development"
"toolkit/2.0:Insulated Development"

   Baseline selection for process rule toolkit/2.0:Collaborative
Development
    Baseline Selection Mode: Latest Baseline Projects
    Prep Allowed:            No
    Versions Matching:
    Release Purposes:

   Baseline selection for process rule toolkit/2.0:Insulated Development
    Baseline Selection Mode: Latest Baseline Projects
    Prep Allowed:            No
    Versions Matching:
    Release Purposes:

   New Projects use process rule toolkit/2.0:Collaborative Development by
default
   New Projects use process rule toolkit/2.0:Insulated Development by
default

   Folder Templates and Folders only in process rule toolkit/
2.0:Collaborative Development
    Template All Completed Tasks for Release %release for Collaborative
projects from Database %database

   Folder Templates and Folders only in process rule toolkit/
2.0:Insulated Development
    Template Integration Tested Tasks for Release %release from Database
%database
    Template Master Integration Tested Tasks for Release %release

   Folder Templates and Folders in both process rules
    Template Assigned Or Completed Tasks for %owner for Release %release
from Database %database
    Template Miscellaneous Tasks for %owner for Release %release
```

Copy a Process Rule

- Copy the `2.0:Insulated Development` process rule over the existing `2.1:Insulated Development` process rule.

```
   ccm process_rule -copy "2.0:Insulated Development" "2.1:Insulated
Development"
```

Delete Process Rules

- Delete the `2.1:Shared` process rule.

```
ccm pr -delete "2.1:Shared"
```

List Current Process Rules

- List all currently defined process rules.

```
ccm process_rule -list

Collaborative Development
Insulated Development
Custom Development
Shared Development
Visible Development
Integration Testing
System Testing
1.0:Integration Testing
1.0:System Testing
1.0:Insulated Development
2.0:Integration Testing
2.0:System Testing
2.0:Collaborative Development
2.0:Insulated Development
Local Collaborative Development
Local Integration Testing
Master Integration Testing
```

Modify a Process Rule

- Set the `2.1:Insulated Development` process rule to use the latest baseline.

```
ccm pr -m "2.1:Insulated Development" -latest_baseline
```

- Set the `2.1:Insulated Development` process rule to use the latest baseline projects with the specified release and purpose combinations.

```
ccm pr -m "2.1:Insulated Development" -latest_baseline_projects -
baseline_release_purpose "2.1:Integration Testing,2.1:System
Testing,2.0:Any"
```

- Edit the list of release/purpose pairs that are used by a specific process rule to search for a baseline.

```
ccm pr -modify -baseline_release_purposes "2.0:Any,1.0:System Testing"
-prepend "2.0:Integration Testing"
```

- Select a baseline named `Build_1234_int` for a process rule whose *`process_rule_spec`* is `2.0:Insulated Development`.

```
ccm process_rule -modify -bn Build_1234_int "2.0:Insulated
Development"
```

Set the Controlling Database for Process Rules

- Set the controlling database to use the `2.1-patch1:Insulated Development` process rule.

```
ccm pr -controlling_database -accept A "2.1-patch1:Insulated
Development"
```

Show Information about Process Rules

- Show all properties of the `2.1:Insulated Development` process rule.

```
ccm process_rule -show info "2.1:Insulated Development"
```

project_grouping command

Synopses

Add Tasks to the Update Properties of a Project Grouping

```
ccm pg|project_grouping
          -at|-add_task|-add_tasks task_spec|all_removed
          project_grouping_spec
```

Copy Tasks from One Project Grouping to Another Project Grouping

```
ccm pg|project_grouping
          -ct|-copy_tasks
          project_grouping_spec1
          project_grouping_spec2
```

Delete a Project Grouping and its Member Projects

```
ccm pg|project_grouping
          -d|-delete [-m|-members|-nm|-no_members]
          project_grouping_spec
```

List Project Groupings

```
ccm pg|project_grouping -l|-list
          [-r|-release release]
          [-purpose purpose_spec]
          [-o|-owner owner_spec]
          [-f|-format "format_string"]
          [-ns|-no_sort]
          [-u|-un_numbered]
          [-sep separator_char]
          [-nf]
```

Refresh the Baseline and Tasks of a Project Grouping

```
ccm pg|project_grouping
          -rbt|-refresh|-refresh_baseline_and_tasks
          project_grouping_spec
```

Remove Tasks from the Update Properties of a Project Grouping

```
ccm pg|project_grouping
          -rt|-remove_task|-remove_tasks task_spec|all|all_added
          project_grouping_spec
```

Set the Auto-refresh Mode of a Project Grouping

```
ccm pg|project_grouping
          -ar|-auto_refresh_baseline_and_tasks|-thaw
          project_grouping_spec
ccm pg|project_grouping
          -no_ar|-no_auto_refresh_baseline_and_tasks|-freeze
          project_grouping_spec
```

Show the Properties of a Project Grouping

```
ccm pg|project_grouping -sh|-show keyword
          project_grouping_spec
          [-f|-format "format_string"]
          [-ns|-no_sort]
          [-u|-un_numbered]
          [-sep separator_char]
          [-nf]

ccm pg|project_grouping -sh|-show
          i|info|information|
          n|name
          r|release|
          p|purpose|
          created_in
          o|owner|
          ar|auto_refresh_baselines_and_tasks
          rtime|refresh_time
          project_grouping_spec
```

Description and uses

Project groupings are used to organize projects by release and purpose for the update operation. A project grouping's task and baseline properties are used when a project is updated so that member selection is consistent across all projects in the group. A project can be a member of only one project grouping. A project grouping is created automatically when a project is created.

Project groupings can be private or non-private. All projects in a private project grouping have the same owner, release, purpose, and state as the project grouping. Private project groupings are identified in one of the following ways:

- `My` *`release purpose`* `Projects` -- the owner of the project grouping is the same as the current user and the database is not DCM-enabled, or the project grouping was created in the local database, such as `My CM/7.1 Insulated Development Projects`.
- *`owner's release purpose`* `Projects` -- the owner of the project grouping is a different user and the database is not DCM-enabled, or the project grouping was

created in the local database, such as `Mary's CM/7.1 Insulated Development Projects`.

- `My` *`release purpose`* `Projects from Database` *`dbid`* -- the owner of the project grouping is the same as the current user and the database is DCM-enabled, and the project grouping was not created in the local database, such as `My CM/7.1 Insulated Development Projects from Database D`.
- *`owner's release purpose`* `Projects from Database` *`dbid`* -- the owner of the project grouping is a different user and the database is DCM-enabled, and the project grouping was not created in the local database, such as `Mary's CM/7.1 Insulated Development Projects from Database D`.

All projects in a non-private project grouping have the same release, purpose, and state as the project grouping. Non-private project groupings are identified in one of the following ways:

- `All` *`release purpose`* `Projects from Database` *`dbid`* for DCM-enabled databases, where *`dbid`* is the database id of the database in which the project grouping was created, such as `All CM/7.1 Integration Testing Projects from Database D`.
- `All` *`release purpose`* `Projects` for non DCM-enabled databases, such as `All CM/7.1 System Testing Projects`.

Every local project grouping is associated with the process rule that corresponds to its release and purpose. A project grouping always has one, and only one, related process rule.

However, note that in some cases, all projects in a project grouping may not have update properties as specified by the project grouping. Those that use process rules will have the same update properties. But a project grouping can contain projects that don't use process rules, or even projects that update using objects instead of tasks. The ability to place them in the same grouping enables you to create baselines from the full set of projects.

In order to have the appropriate update properties, project groupings have many associations with other objects in the database. Because process rules use folders and tasks, these same folders and tasks are associated with a project grouping that use process rules. In addition, a project grouping has a set of saved tasks, a set of additional tasks, a set of removed tasks, and a set of automatic tasks, each of which is specific to the project grouping. You can also add and remove tasks in the grouping. Every local project grouping also has a relationship to a baseline, if the process rules use baselines.

For more detailed information about how build managers can best use project groupings, see the *Build Manager's Guide.*

Use the `project_grouping` command to:

- Show information or associated projects, objects, and tasks for a specific project grouping.

- List project groupings.
- Modify a project grouping.
- Delete an existing project grouping.

After a user has created a project, the user can refresh baselines and tasks, add and remove tasks, update the grouping, copy tasks to the grouping, specify auto-refresh options, and delete the grouping.

Options and arguments

project_grouping_name

The *project_grouping_name* is the name that is assigned to the project grouping.

-at|add_task|add_tasks

Adds the specified task or tasks to the project grouping.

If a task is in Removed Tasks, it is removed from Removed Tasks and added to Saved Tasks. In this case, required tasks on which the specified task depends are not added, regardless of the setting of the include_required_tasks option.

If a task is in neither Removed Tasks nor Saved Tasks, it is added to Added Tasks. If the include_required_tasks option is TRUE, required tasks are computed and added to Added Tasks as well.

If the string all_removed is specified in the *task_spec*, all removed tasks are added back, and required tasks are not computed.

If you cancel this command while in progress, be sure to read the message about the status of the operation. The outcome will vary, depending on whether any tasks have been added before you cancelled.

-ar|-auto_refresh_baselines_and_tasks|-thaw

Specifies that the project grouping always refreshes the baseline and tasks during an update operation.

-ct|-copy_tasks *project_grouping_spec1 project_grouping_spec2*

Copies the net tasks (Saved Tasks plus Added Tasks) from one project grouping to another. The tasks are added to the second project grouping in the same way as if the -add_tasks option had been used.

However, dependency analysis is not done, and required tasks are not calculated. This gives you a way to add the exact set of tasks to a different project grouping.

-delete

Deletes the project grouping that has the *project_grouping_name* you specify.

If -no members is specified, or -members is not specified, only the project grouping is deleted if it does not have any projects in it. If is does have projects in it, the command will fail.

If -members is specified, the project grouping is deleted along with all associated projects, and all associated folders that are not used in any project or project grouping are also deleted.

-description "*project_grouping_description*"

Optionally, provides a detailed description of the project grouping. There is no limit on its length, and no restriction on its contents. The *project_grouping_description* must be enclosed in double quotes if it contains one or more spaces.

-f|-format "*format_string*"

Specifies the command's output format. The default format depends on the options that you use with -format (for example, -list or -show) and those options' keyword arguments. To learn more about the default output formats, see the descriptions of the options that you can use with -format.

The format can contain a combination of text and keywords. Keywords are replaced by specific data about each object. For example, the keyword %owner is replaced with sue if information about an object owned by user sue is displayed.

-list

Lists all project groupings in the database. If -release, -purpose, or -owner is specified, the list shall be limited to those project groupings whose properties match those specified.

The default format is `%displayname`, but it can be overridden by the specification of the `-format` option.

This command sets the selection set to the list of project groupings returned.

`-nf`

Specifies that the output will not be displayed in columnar format.

`-no_ar|-no_auto_refresh_baselines_and_tasks|-freeze`

Specifies that the project grouping always uses the saved baseline and tasks.

`-ns|-no_sort`

Specifies that the command's output will not be sorted.

`-purpose` *purpose_spec*

Specifies the purpose of the projects that are to be listed.

`-rbt|-refresh_baseline_and_tasks`

Specifies that the baseline and tasks be refreshed and saved on the project grouping, regardless of the project grouping's auto-refresh mode. Even if the project grouping is frozen, the baseline and tasks are updated. However, it does not change the auto-refresh setting of the project grouping.

`-rt|-remove_task|-remove_tasks` *task_spec*`|all|all_added`

Removes a task or tasks from a project grouping.

If the task is in `Added Tasks`, it is removed from `Added Tasks`. If the task is in `Saved Tasks`, it is removed from `Saved Tasks` and added to `Removed Tasks`. In no cases will dependent tasks be calculated and removed.

If `all` is specified, all `Added Tasks` are removed, and all `Saved Tasks` are added to the `Removed Tasks` folder. The remove task operation is applied to each `Added Task` and each `Saved Task`.

It `all_added` is specified, all tasks are removed from `Added Tasks`. The remove tasks operation is applied to each Added Task.

`-sep` *`separator_char`*

Used only with the `-format` option. Specifies the separation character you are using in your *`format string`* to separate input fields from one another. The command output displays your fields in columns, separated by spaces, wherever you specify this character.

`-show`

Shows the properties associated with the *`project_grouping_name`* you specify.

`i|info|information`

If specified, `name`, `release`, `purpose`, `owner`, and `created_in` information is displayed.

`n|name`

If specified, the project grouping's name is displayed.

`r|release`

If specified, the project grouping's release value is displayed.

`p|purpose`

If specified, the project grouping's purpose is displayed.

`o|owner`

If specified, the name of the project grouping's owner is displayed.

`created_in`

If specified, the name of the database where the project grouping was created is displayed.

`rtime|refresh_time`

If specified, the time that the baseline and tasks were last computed (and saved) is displayed.

-- The following keywords are supported with -show:

proj|projects

If specified, all projects that are included in the project grouping are displayed. The default format is:

%displayname %status %owner %release %create_time

The default format may be overridden by using the -format option.

bl|baseline

If specified, the baseline name is displayed. The default format is:

%displayname: %description

fo|folders

If specified, the project grouping's folders are displayed. The default format is:

%displayname

ar|auto_refresh_baselines_and_tasks

If specified, the auto-refresh mode of the project grouping is displayed. The auto-refresh is either TRUE, (enabled or thawed), or FALSE (disabled or frozen).

st|saved_tasks

If specified, all tasks that are in the project grouping's Saved Tasks are displayed. The Saved Tasks are the tasks that are shown in the Rational Synergy **Project Grouping Properties** dialog, in the **Baseline & Tasks** tab, under the heading "Tasks on top of the baseline". These are the tasks determined from the project grouping's process rule.

The default format is:

%displayname %release %owner %create_time

The default format may be overridden by using the -format option.

at|added_tasks

If specified, all tasks that are in the project grouping's Added Tasks are displayed. Added Tasks are the tasks that the user added manually. These are the tasks that are shown in the Rational Synergy **Project Grouping Properties** dialog, in the **Manually Added Tasks** tab.

The default format is:

`%displayname %release %owner %create_time`

The default format may be overridden by using the `-format` option.

`rt|removed_tasks`

If specified, all tasks that are in the project grouping's Removed Tasks are displayed. The Removed Tasks are those tasks in the Rational Synergy **Baseline & Tasks** tab that have a cleared checkbox. These are the tasks that the user manually removed.

The default format is:

`%displayname %release %owner %create_time`

The default format may be overridden by using the `-format` option.

`all_tasks`

If specified, the net tasks (Saved Tasks plus Added Tasks) are displayed. The default format is:

`%displayname %release %owner %create_time`

The default format may be overridden by using the `-format` option.

`obj|objs|objects`

If specified, all objects that are included in all projects in the project grouping are displayed. The default format is:

`%displayname %status %owner %release %create_time`

The default format may be overridden by using the `-format` option.

`automatic_tasks`

If specified, the automatic tasks from the process rule that are consequently related to the project grouping are displayed. The default format is

`%displayname %release %owner %create_time`

The default format may be overridden by using the `-format` option.

`-u|-un_numbered`

Suppresses automatic numbering of the command's output (that is, the output is un-numbered).

Examples

- Refresh the baseline and tasks of a project grouping named `My CM/6.3 Collaborative Development`.

  ```
  ccm project_grouping -refresh "My CM/6.3 Collaborative Development"
  ```

- Show information about a project grouping named `My Fav/5.1 Visible Projects`.

  ```
  ccm pg -show info "My Fav/5.1 Visible Projects"
  ```

Related topics

- update_properties command
- process_rule command

project_purpose command

Synopsis

Create a Project Purpose

```
ccm project_purpose -cr|-create -n|-name "purpose name"
                    -stat|-status status
                    [-ms|-member_status member_status]
```

Delete a Project Purpose

```
ccm project_purpose -d|-delete [-y] purpose_specs
```

Modify a Project Purpose

```
ccm project_purpose -m|-modify [-n|-name "new name"]
                    [-ms|-member_status new_member_status] purpose_spec
```

Show a Project Purpose

```
ccm project_purpose -s|-sh|-show [-stat|-status status]
                    [-r|-role [role]] [purpose_spec]
```

Description and uses

The `project_purpose` command enables you to create, delete, modify, or show (depending on your user role) the project purposes for a Rational Synergy database. The project purposes are used to set up multiple *prep*, *shared*, *working*, or *visible* versions of the same project for different uses, such as different levels of testing.

The project purposes include the following:

- Purpose name

  This name reflects the purpose, for example, performance testing, personal use, etc.

- Member status for the purpose

  The member status enables you to differentiate projects of the same state being used for different purposes when you perform an update operation. For example, you could define three unique levels of system testing called `sqa1`, `sqa2`, and `sqa3`.

- Status of the project

  The status shows what state projects (*working*, *prep*, etc.) of this purpose can use.

All users can show project purposes. A project purpose manager can create or delete a project purpose. You must be in the *ccm_admin* role to edit a project purpose.

The project purpose table affects the following:

- Options that are displayed in the Purpose field for the following dialog boxes: **Copy Project, Create Project, Properties** (for a project or product)
- Options you can specify in the following commands: `ccm copy_project`, `ccm create`, `ccm create -type project`, and `ccm properties`
- Specifies the `status` and `member_status` values that are used for the projects copied using each purpose option
- Determines which automatic tasks projects will be associated with
- Affects the synopses of corresponding automatic tasks

Each Rational Synergy database contains one project purpose list only. You can define project purpose lists for each release.

The default subsets of the project purpose table define the following purposes:

```
Integration Testing:       prep:         integrate
System Testing:            prep:         sqa
Insulated Development:     working:      working
Collaborative Development: working:      collaborative
Shared Development:        shared:       shared
Visible Development:       visible:      visible
```

Additionally, if your database is DCM-enabled, it will also have the following purpose:

```
Master Integration Testing:  master_integrate:    prep
```

This project purpose table defines one purpose per line. The line has the following format:

```
purpose_label:   status:   member_status
```

`purpose_label` is the option that is displayed in the dialogs' Purpose fields and the purpose you would specify with the commands' -purpose options.

`status` is the state of any project with that purpose.

`member_status` is the value that your new project's `member_status` attribute will have. The `member_status` value should be similar to the purpose. For example, if you create a purpose of `Test Integration`, the member status should be set to a similar value, such as `test_int`. This value can be used in place of the purpose in CLI commands.

If you update using object status, all of your `member_status` values must be actual states defined in your database. If you update using tasks, your `member_status` values can be any unique word that will differentiate your *prep* or *shared* projects from one another when you update. For example, you could set this option to the following value to define four unique levels of system testing:

```
Insulated Development:    working:     working
Integration Testing:      prep:        integrate
System Testing 1:         prep:        sqa1
System Testing 2:         prep:        sqa2
System Testing 3:         prep:        sqa3
System Testing 4:         prep:        sqa4
```

Options and arguments

`-cr|-create`

Creates a project purpose. The `-name`, "*`purpose name`*", and `-status` options are required with this option.

Also, you can specify the `-member_status` option if you want a specific value. If you do not specify a member status, the `member_status` value will default to the value of the `-status` option you entered.

`-d|-delete` *`purpose_spec`*

Deletes a project purpose specified by *`purpose_spec`*.

Once a project purpose is deleted, the following behavior will occur:

- Projects cannot be checked out or created with the deleted purpose.
- Existing projects and products retain the member status setting.
- Existing projects and products cannot have their purpose changed to the deleted purpose.
- Process rules for that purpose are deleted.

`-m|-modify`

Modifies a project purpose. When used with the `-name` *`purpose_spec`* options, specifies the name of the project purpose to change. When used with the `-member_status` option, specifies the value of the member status to change.

`-ms|-member_status`

Specifies a project purpose's member status. The member status enables you to differentiate projects of the same state being used for different purposes when you update. The value must be unique in the database.

Use this option with the `-create` option to specify the member status of the purpose you are creating; use it with the `-modify` option to specify the member status of the purpose you are modifying. Values from the project purpose table are used for each purpose option.

The purpose and the member status should be similar. For example, if you are creating a purpose of `Test Integration`, then the member status should be set to a similar value, such as `test_int`.

`-n|-name` *"purpose name"*

Names a project purpose. Use this option with the `-create` option to specify the name of the project purpose you are creating; use it with the `-modify` option to specify the new name of a project purpose you are modifying.

purpose_spec

Is either *"purpose name"* or `member_status`.

When used with the `-modify` option, specifies the project purpose to change.

purpose_specs

Specifies that you will use more than one *purpose_spec*. There must be at least one space between *purpose_spec*s.

When used with the `-show` option, displays the project purposes specified by *purpose_specs*.

`-r|-role` [*role*]

When used with the `-show` option but without specifying a role, displays the purposes that can be used by the role you are in. When used with the `-show` option and a specified role, displays the purposes that can be used by the specified role.

`-s|-sh|-show` [*purpose_specs*]

Shows project purposes. When used without the *purpose_specs* option, displays all project purposes. When used with the *purpose_specs* option, displays the project purposes specified by *purpose_specs*.

`-stat|-status` *status*

Specifies the status. When used with the `-show` option, specifies that you want to view project purposes with a specific status setting. When used with the `-create` option, specifies the status for the project purpose. Values from the `project_purpose_list` are used for each purpose option.

`-y`

When used with the `-delete` option, deletes the project purpose without displaying confirmation messages.

Examples

- Create a project purpose with a name of `Test Purpose`, a status of *prep*, and a member status of `test`. View the newly created purpose.

```
ccm project_purpose -cr -name "Test Purpose" -stat prep -ms test
ccm project_purpose -s "Test Purpose"
Purpose         Member Status     Status
 Test Purpose    test              prep
```

- Delete a project purpose called `Test2 Purpose`.

```
ccm project_purpose -d "Test2 Purpose"
Are you sure that you want to delete purpose 'Test2 Purpose'? (Yes/No)
[No] Yes
```

- Change a project purpose's name and member status.

```
ccm project_purpose -m -n "Test2 Purpose" -ms test2 "Test Purpose"
```

- Show the project purposes for a user in the *developer* role.

```
ccm set role developer
ccm project_purpose -s -r
 Purpose                Member Status     Status
 Insulated Development   working           working
 Shared Development      shared            shared
 Visible Development     visible           visible
 Collaborative Development
```

properties command

Synopsis

Show Properties

```
ccm prop|properties|info [-s] [-n] file_spec [file_spec...]
ccm prop|properties|info [-s] [-n] -p |
                         -project project_spec [project_spec...]
```

Show an Arbitrary Set of Attributes

```
ccm prop|properties|info [-sep separator_char] [-nf]
                         -f|-format "format_string" file_spec [file_spec...]
ccm prop|properties|info [-sep separator_char] [-nf]
                         -f|-format "format_string"
                         -p|-project project_spec [project_spec...]
```

Description and uses

The `properties` command provides information about one or more objects.

This command displays the attribute values of a group of model-defined attributes for the specified object(s) to standard output.

If you are using object status to configure your projects, the `exclude_status` and `member_status` attributes are displayed from the command line; otherwise, the `baseline` attribute is shown.

For additional information about using the `ccm properties` command to show information about object versions, see Selection set reference form.

Options and arguments

`file_spec`

Specifies the name of the file or directory for which information is being displayed.

`-f|-format "`*`format_string`*`"`

Specifies the output format of the properties.

You can specify keywords or rearrange the keywords. For example, you can specify just `%objectname`, which will return the query information in the object reference form (that is, *`object_name-version`*`:`*`type`*`:`*`instance`*) or use `%displayname` to return the query information without type and instance (that is, *`object_name-version`*).

The required string uses keywords and literal text, such as:

```
%displayname owned by
```

A keyword can be built-in (`%fullname`, `%displayname`, `%objectname`) or the name of any existing attribute such as `%modify_time` or `%status`.

`-n`

Do not display labels (in default mode) or a header (single-line mode) when displaying an object's properties.

You cannot use this option if you are using the `-f` option.

`-nf`

Do not make the output columnar.

You can use this option only with the `-f` option.

`-p` *`project_spec`*

Specifies the name of the project for which information is to be displayed.

`-s`

Shows the information on a single line.

`-sep` *`separator_char`*

Used only with the `-format` option, specifies the separation character you are using in your *`format_string`* to separate input fields from one another. Rational Synergy displays your fields in columns, separated by spaces, wherever you specify this character.

Note that `-sep` has no affect on output. The columns are always space-separated unless you use `-nf` to suppress columns altogether.

Examples

- Obtain information on the `os_ico-1` project, which uses object status to update.

```
ccm prop -p os_ico-jeff
name           : os_ico
version        : john
owner          : john
status         : working
type           : project
create_time    : Tue Aug 13 11:09:33 1998
modify_time    : Thu Aug 15 10:41:35 1998
platform       : <void>
release        : 2.1
task           : 6
member_status  : working
exclude_status : public shared visible
```

  Note that the project has `member_status` and `exclude_status` attributes.

- Obtain information on the `task_ico-2` project, which uses tasks to update.

```
ccm prop -p task_ico-2
name           : task_ico
version        : 2
owner          : sue
status         : working
type           : project
create_time    : Tue Aug 13 11:09:33 1998
modify_time    : Thu Aug 15 10:41:35 1998
platform       : <void>
release        : 4.6
task          : 6
```

- Show the release values of all of the objects in the current directory.

```
ccm prop -f "%objectname %release" *
```

query command

Synopsis

```
ccm query [-ns|-no_sort]


ccm query [-n|-name name] [-ns|-no_sort]
          [-o|-owner owner] [-s|-state state]
          [-t|-type type] [-v|-version version]
          [-i|-instance instance]
          [[-task task_number] [-db database_ID]]
          [-f|-format "format_string"] [-u] [-nf]
          [-sep separator_char] ["query_expression"]
```

Description and uses

Use the `query` command to search for objects in the database by evaluating a query expression. The query results are displayed sorted unless you specify no sort.

> **Note** If a query function provides a sorting capability, and if you combine that query function with other query operators to make a compound query, the sorting will be lost.

The results of the query are placed in the selection set, then are available for use as arguments to subsequent commands by using the selection set reference syntax ("*@listed_object_number*").

The `-db` option is available only if your database is initialized for DCM.

Options and arguments

`-db` *database_ID*

This option can only be used with the `-task` option. When specified, it finds the objects that are associated with the specified task that were created in the specified database. The default is to find objects associated with the task created in any database.

`-f|-format "`*format_string*`"`

Specifies the output format of the query.

The default format is:

```
%displayname %status %owner %type %project %instance %task
```

where the `%displayname` keyword is made up of the name and version keywords, separated by the default delimiter, `%name-%version`.

You can specify other keywords or rearrange the keywords. For example, you can specify just `%objectname`, which will return the query information in the object reference form (that is, *`object_name-version`*`:`*`type`*`:`*`instance`*), or use `%displayname` to return the query information without type and instance (that is, *`object_name-version`*).

The required string may use keywords and literal text, such as:

```
%object %displayname is owned by %owner
```

A keyword can be built-in (`%fullname`, `%displayname`, `%objectname`) or the name of any existing attribute or pseudo-attribute such as `%modify_time` or `%status`.

See Built-In keywords for a list of keywords.

`-i|-instance` *`instance`*

Finds all objects with the instance number *`instance`*.

`-n|-name` *`name`*

Finds all objects named *`name`*.

`-nf`

Do not make the output columnar.

`-ns|-no_sort`

Do not sort the output.

`-o|-owner` *`owner`*

Finds all objects owned by *`owner`*.

"*`query_expression`*"

Specifies the expression for the query. For information on query expressions, see Query expressions.

`-s|-state` *`state`*

Finds all objects in the *`state`* state.

`-sep` *`separator_char`*

Used only with the `-format` option. Specifies the separation character you are using in your *`format_string`* to separate input fields from one another. Rational Synergy displays your fields in columns, separated by spaces, wherever you specify this character.

`-task` *`task_number`*

Finds only the objects associated with the specified task.

`-t|-type` *`type`*

Finds all objects of type *`type`*.

`-u`

Suppresses automatic numbering of this command's output ("un-numbered").

`-v|-version` *`version`*

Finds all objects with the version *`version`*.

Examples

- List all objects named `foo.c` owned by *mary*.

```
ccm query -n foo.c -o mary
1) foo.c-1    integrate bob nasub1 csrc 1 1
2) foo.c-1.2  working   bob nasub1 csrc 1 4
3) foo.c-2    working    bob nasub2 csrc 1 5
```

- To look at the source contents of item 3 in the selection set, enter the following.

```
ccm cat @3
```

- List all objects named `foo.c`, owned by *sue*, for task 4.

```
ccm query -n foo.c -o sue -task 4
1) foo.c-1.2  working  sue csrc 1 4
```

- List the name and time last modified of all objects named `brochure.doc` owned by *fiona*.

```
ccm query -n brochure.doc -o linda -f "%name %modify_time"
1) brochure.doc Tue Aug  6 12:17:55 1996
```

- List all objects associated with task 3 that are from the `santa_fe` database.

  ```
  ccm query -task 3 -db santa_fe
  1) DropEdit.cpp-1    integrate tom c++ diffmerge santa_fe#1 <void>
  2) vdifmrgDoc.cpp-1  integrate tom c++ diffmerge santa_fe#1 <void>
  ```

- List change requests associated with a particular transfer set.

  ```
  ccm query query_expression
  ```

  where *`query_expression`* is the change request query that is being used for the transfer set, and includes "`cvtype=problem`".

  For example:

  ```
  ccm query "cvtype='problem' and product_name='myproduct'"
  ```

- Show release-specific process rules that are instantiations of the "Collaborative Development" generic process rule.

  ```
  ccm query ''cvtype='process_rule' and name='Collaborative
  Development'' -f "%none %is_generic_pr_of"
  ```

Related topics

- finduse command

reconcile command

Synopsis

```
ccm rwa|reconc|reconcile]
              [-t|-task task]
              [-c|-comment string]
              [-r|-recurse|-no_recurse]
              [-iu|-ignore_uncontrolled|-cu|-consider_uncontrolled]
              [-mwaf|-missing_wa_file|-imwaf|-ignore_missing_wa_file]
              [-if file_spec]
              [-rpt|-report file_spec]
              [-s|-show]
              [-udb|-update_db|-uwa|-update_wa]
              [-p|-project] project_spec [project_spec...]
```

Description and uses

The `reconcile` command compares the files in your work area with your database files. If reconcile is able to resolve differences automatically, it does so. Otherwise, it identifies the files as a conflict and takes no action. Reconcile will update files automatically in the following cases:

- If you have a file checked out and you changed the file in your work area, the database is updated automatically. This does not occur when you use the `-update_wa` option, which updates the work area from the database.
- If you have a file checked out and have updated the database from another work area, this work area is updated automatically.

When the reconcile operation is unable to update either the database or the work area automatically, it identifies the file as being in conflict. Conflicts occur in the following cases:

- You modified a file in your work area, whether or not it was checked out.
- You changed the database copy of a file from another work area and you changed the same file in this work area.
- You changed a file in the database, but the work area being updated was not available to update.
- You created a file in the work area, but did not place it under source control.
- You checked in a file from another work area, but the work area was not available to update with changes.
- You removed a file from the work area, but did not delete it from your project.

Additional errors can occur with controlled links and symbolic links and the work area paths. You will need to manually resolve these types of conflicts. For additional information about work area conflicts, see Work area conflicts.

A few other ways to use this command with files that are checked out include:

- If your work area is on a laptop and you are able to work disconnected from Rational Synergy, you can use the `reconcile` command to bring your work area and the database back in sync.
- On UNIX, If a tool you are using breaks the link(s) between an object(s) you are modifying and the Rational Synergy database, the `reconcile` command will reconcile the changes, then re-establish the link(s).

For example, if you do not have a Rational Synergy session up and you need to modify an object that is not checked out, you can change it in your work area then update the Rational Synergy database later. Do this by resetting the Read Only attribute on the file and modifying it. Later, when you bring up a Rational Synergy session, you can use the `reconcile` command to update your database with the work area changes.

> **Note** To stop a reconcile from the CLI, enter `<CTRL+C>` at any time.
>
> When you stop the reconcile from the CLI, you will receive a message stating that errors may occur in your work area. The errors will not occur until you try to use the work area; to avoid problems, reconcile the work area completely before you use it.

Options and arguments

`-c|-comment` *`string`*

Specifies the comment string.

`-cu|-consider_uncontrolled`

Specifies to consider uncontrolled files during reconcile. During the reconcile process, the work area is checked for files that are not under source control. When uncontrolled files are found, they are automatically created in the database and placed under control if the `-update_db` option is also specified. The file extension is used to determine the type.

If neither `-consider_uncontrolled` or `-ignore_uncontrolled` is specified, the default is to ignore uncontrolled files

`-if|-ignore_types`

Specifies not to reconcile files whose file name contains the specified extension. This option works only for uncontrolled files, and must be used with the `-consider_uncontrolled` option.

Use a comma, semicolon or white space to separate entries.

`-imwf|-ignore_missing_wa_file`

Specifies not to reconcile files that are missing in the work area. Use this option when you don't want all the files from a project in your work area.

`-iu|-ignore_uncontrolled`

Specifies not to reconcile files that are uncontrolled. If neither `-ignore_uncontrolled` or `-consider_uncontrolled` is specified, the default is to ignore uncontrolled files.

`-mwaf|-missing_wa_file`

Compares the database against the work area, looking for missing work area files. During the reconcile process, files in the work area are checked against those in the database. When a file is found in the database but not in the work area, it is copied to the work area if the `-update_wa` option is specified. Use this option if you want to ensure that you have the same files in your database and your work area. However, this can take some time on large projects.

`-nr|-no_recurse`

Specifies not to reconcile subprojects belonging to the project you specify for reconciliation.

`-p|-project` *`project_spec`*

Specifies the project to be reconciled.

`-rpt|-report`

Generates a text report about the reconcile process.

`-r|-recurse`

Specifies to reconcile subprojects belonging to the project you specify for reconciliation.

This option controls the depth of a reconcile operation when you synchronize a project. This is important because if you are synchronizing a top-level project with many nested subprojects, a recurse reconcile could take a substantial amount of time and resources. You should carefully choose whether to recurse as it will reconcile every subproject beneath your specified top-level project. If you do not synchronize the hierarchy, you will save time and resources. Alternatively, if you need to reconcile the entire hierarchy, this option enables you to do so in one operation.

`-s|-show`

Shows the conflicts without resolving them. This is the default.

`-t|-task` *task*

Associates any objects being checked out with the specified task.

If the current (default) task is set and you do not specify a different task, the objects you are checking out are associated with the current task automatically.

`-udb|-update_db`

Updates the database with versions in your work area. Uses of this option include:

- If you modified a file that was not checked out, reconcile creates a new version by default, and the database is updated with your changes.
- If you updated the database copy of a file from another work area and you changed the same file from this work area, reconcile updates the database from this work area.

Use this option when you are certain that the work area represents the correct set of changes.

`-uwa|-update_wa`

Updates your work area with versions from your database. Use this option when you are certain that the database represents the correct set of changes

Examples

- Reconcile the file `foo.c` by updating the database from the work area.

  ```
  ccm reconcile -update_db foo.c-1:csrc:1
  ```

- Reconcile the `ico_june16-1` project, but do not reconcile files whose file name contains any of the following extensions: `.doc`, `.gif`, or `.exe`.

  ```
  ccm reconcile -p ico_june16-1 -ignore_types "*.doc;*.gif;*.exe"
  ```

- Reconcile the directory `src` in `proj1`, update the work area from database, and check for missing files.

  Windows: `ccm reconcile -missing_wa_file -update_wa c:\users\bhoskins\ccm_wa\proj1-1\src`

  UNIX: `ccm reconcile -missing_wa_file -update_wa /users/bhoskins/ccm_wa/proj1-1/src`

- Reconcile the project `proj1` and subprojects, updating the database from the work area, checking for uncontrolled files.

```
ccm reconcile -recurse -consider_uncontrolled -update_db -project
proj1-1
```

- Reconcile the project `proj1` and subprojects without updating the database, ignore `*.tmp` files, and generate a report.

```
ccm reconcile -recurse -report conflict.txt -if *.tmp -project proj1-1
```

A portion of a sample report is shown below.

```
Reconcile Report: 06/24/99 10:02:06

Project:
   proj1-1

Options:
   Conflict Handling: Select
   Ignore Files: *.tmp
   Generate Report: c:\temp\conflict.txt
   Recurse Hierarchy

Conflict Summary:
   34 Project(s) reconciled
   129 Directories reconciled
   1 Work Area Change(s) to Working Object(s)
   1 File(s) Missing from Work Area
   105 Uncontrolled File(s)
   2 File(s) Ignored
   235 File(s) not in Conflict
-------------------------------------------------------------------------
   344 Total Files Examined

Conflict Details:
1 Work Area Change(s) to Working Object(s)
-------------------------------------------------------------------------
Conflict:        Work Area file: 'file.c-1' does not match the Database
File:            d:\users\joe\ccm_wa\ccm46\proj1-1\proj1\x1\file.c
File Type:       Source File
Project Name:    proj1-1
Object Name:     file.c-1
Object Status:   working
Work Area Time:  06/23/98 14:34:04
Database Time:   06/19/98 12:39:33


1 File(s) Missing from Work Area
```

```
-------------------------------------------------------------------------
--
Conflict:        'file.h-1' is missing from the Work Area
File:            d:\users\joe\ccm_wa\ccm46\proj1-1\proj1\x1\file.h
File Type:       Source File
Project Name:    proj1-1
Object Name:     file.h-1
```

- UNIX: Reconcile the `ico_june16-1` project and reconcile its subprojects.

  This example assumes that you needed to modify four objects outside of the Rational Synergy product. You made copies of the four objects in your work area so that you could modify the objects. Two of the objects were in the *working* state: `bufcolor.c` and `clear.c`. Two of the objects were in the *integrate* state: `drawbuf.c` and `concat.c`. After you modified these files, you reconciled the project from the command line.

  ```
  ccm reconcile -p ico_june16-1
  Work area reconciliation starting...
  recursing hierarchy, conflicts will be automatically updated
  Updating '/users/linda/ccm_wa/ccmint15/ico_june16-1'...
  Updating database with file '/users/linda/ccm_wa/ccmint15/ico_june16-
  1/ico_june16/src/bufcolor.c'...
  Updating database with file '/users/linda/ccm_wa/ccmint15/ico_june16-
  1/ico_june16/src/clear.c'...
  Creating new members for project ico_june16-1 ...
  Creating version 2 of concat.c-1:csrc:1 ...
  Updating database with file '/users/linda/ccm_wa/ccmint15/ico_june16-
  1/ico_june16/src/concat.c'...
  Creating version 2 of drawbuf.c-1:csrc:1 ...
  Updating database with file '/users/linda/ccm_wa/ccmint15/ico_june16-
  1/ico_june16/src/drawbuf.c'...
  2 new OV(s) successfully created.
  concat.c-2:csrc:1
  drawbuf.c-2:csrc:1
  Reconciliation complete.
  ```

- UNIX: Reconcile the `ico_june16-1` project, but discard the updates made in your work area and do not reconcile subprojects belonging to the project.

  For this example, assume you were tasked to update the `move.c` object, which was in the *working* state, and the `colname.c` object, which was in the *integrate* state. After you copied and modified these objects in your work area, the direction of the project changed and you ended up not needing these changes after all.

  ```
  % cd ~linda/ccm_wa/ccmint15
  % ls
  ico_june16-1
  ```

```
$ ccm reconcile -p ico_june16-1 -no_recurse
Examining work area for conflicts...
not recursing hierarchy, conflicts will be automatically discarded
Updating '/users/linda/ccm_wa/ccmint15/ico_june16-1'...
Discarding changes to '/users/linda/ccm_wa/ccmint15/ico_june16-1/
ico_june16/src/colname.c'..
Discarding changes to '/users/linda/ccm_wa/ccmint15/ico_june16-1/
ico_june16/src/move.c'...
Reconciliation complete.
```

Note that the work area was updated with the original files from the database, and that the changes made to `colname.c` and `move.c` were discarded.

Related topics

- resync command
- sync command
- work_area command
- Work area conflicts

reconfigure command

The `reconfigure` command is an alias for the update command.

reconfigure_properties command

The `reconfigure_properties` command is an alias for the update_properties command.

reconfigure_template command

The `reconfigure_template` command is an alias for the process_rule command.

relate command

Synopsis

```
ccm relate -s|-show [-l]
           [-l|
           -fmt|-format "format_for_from_object::format_for_to_object"]
           [n|-name relation_name]
           [-f|-from file_spec1] [-t|-to file_spec2]
ccm relate -n|-name relation_name
           -f|-from file_spec1 -t|-to file_spec2
```

Description and uses

The `relate` command enables you to add a relationship (*`relation_name`*) between *`file_spec1`* and *`file_spec2`*, or to show the relationship with the specified data.

More relationships are predefined in Rational Synergy. See Relationships for a table showing these relationships. However, you can define new relationships using the `relate` command.

Options and arguments

`-f|-from` *`file_spec1`*

Specifies the object from which to show or create a relationship.

`-fmt|-format "`*`format_for_from_object`*`::`*`format_for_to_object`*`"`

Specifies the output format for the `-s|-show` option.

The default `relate` command output is as follows:

`'from'_object_info relationship_name 'to'_object_info rel_create_time`

The default format for each object is as follows:

```
%displayname %status %owner %type %project %instance %task
```

where the `%displayname` keyword is made up of the name and version keywords, separated by the default delimiter, `%name-%version`. You can specify other keywords or rearrange the keywords.

The required string may use keywords and literal text, such as:

```
%object %displayname is owned by %owner
```

A keyword can be built in (`%fullname`, `%displayname`, `%objectname`) or the name of any existing attribute such as `%modify_time` or `%status`.

See Built-In keywords for a list of keywords.

You can specify different formats for each object by using the "*format_for_'from'_object*::*format_for_'to'_object*" argument.

For example, to specify a format that shows a description of each folder ('from' object) that has the `task_in_folder` relationship, and the task synopsis for each task ('to' object) in the folder, use the following command:

```
ccm relate -s -n task_in_folder -fmt "%description::%task_synopsis
```

The output would be as follows for each folder:

folder_object_description task_in_folder *task_synopsis*

Do not use this option with the `-l` option. The `-l` option is equivalent to specifying the following:

```
ccm relate -show -sep / -fmt "(%objectname/%status/%owner)" -f
Makefile-2:makefile:17
(Makefile-2:makefile:17 test steveh) successor (Makefile-3:makefile:17
working sue) Fri Nov 19 10:09:13 2004
ccm relate -sep / -fmt "(%objectname/%status/%owner)"
```

`-l`

When showing relationships, use the long output format.

`-n|-name` *relation_name*

Specifies the name of the relationship to create or show.

`-s|-show`

Show the relationships among the specified objects.

`-sep` *separator_character*

Used only with the `-format` option. Specifies the separation character you are using in your *format_string* to separate input fields from one another. Rational Synergy

displays your fields in columns, separated by spaces, wherever you specify this character.

`-t|-to` *`file_spec2`*

Specifies the object to which to show or create a relationship.

Examples

- Make `clear-2` a successor to `clear-1`.

  ```
  ccm relate -n successor -f clear-1 -t clear-2
  ```

- Show the relationship(s) between `clear-2` and `clear-1`.

  ```
  ccm relate -s -f clear-1 -t clear-2
  ```

- Show every relationship in the database from `clear-1` to any other object version.

  ```
  ccm relate -s -f clear-1
  ```

- Link version 5.1.1 of `print.c` to version 6.

  ```
  ccm relate -name successor -from print.c-5.1.1:csrc:1 -to print.c-6:csrc:1
  ```

Related topics

- history command
- unrelate command

release command

Synopsis

Controlling Database

```
ccm release  -cdb|controlling_database
             [-local|-handover dbid| -accept dbid]
             [-component componentname | releasename]
```

Create

```
ccm release  -c|-create [-from releasename] releasename
             [-baseline|-bl releasename]
             [-description|-desc description |
                 -description_file|-desc_file description_file]
             [-manager manager]
             [-active|-inactive]
             [-allow_dcm_transfer|-noallow_dcm_transfer]
             [-allow_parallel_check_out|-noallow_parallel_check_out]
             [-allow_parallel_check_in|-noallow_parallel_check_in]
             [-groups groups]
             [-included_releases included_releases|
                  -included_releases_file included_releases_file]
             [-purposes purpose_spec |
                  -purposes_file purposes_file]
             [-phase phase]
             [-process process_name]
```

Delete

```
ccm release -d|-delete releasename [-force]
            [-old] releasename [[-force]
```

Delimiter

```
ccm release -delimiter [-preview] old_delimiter new_delimiter
```

List

```
ccm release -l|-list [-active|-inactive] [-component componentname]
            [-old]
```

Modify

```
ccm release  -m|-modify releasename
            [-baseline|-bl releasename]
            [-description|-desc description|
               -description_file|-desc_file description_file]
            [-manager manager]
            [-active|-inactive]
            [-allow_dcm_transfer|-noallow_dcm_transfer]
            [-allow_parallel_check_out|-noallow_parallel_check_out]
            [-allow_parallel_check_in|-noallow_parallel_check_in]
            [-groups groups]
            [-included_releases included_releases|
               -included_releases_file included_releases_file]
            [-purposes purpose_spec |
               -purposes_file purposes_file]
            [-phase phase]
```

Rename

```
ccm release -rename oldreleasename newreleasename
            [-all|-local|-dbid dbids|-database_id dbids]
            [-force]
            [-preview]
            [-nocheck]
```

Show

```
ccm release -s|-show (information|active|allow_dcm_transfer|baseline|
               create_time|description|groups|included_releases|
               manager|modifiable_in|owner|parallel_check_out|
               parallel_check_in|phase|phase_log|purposes) releasename
```

Description and uses

Use the `release` command to create, modify, delete, and show release information.

You must be working in the required role to perform a release operation:

- Any user can show or list releases.
- A build manager or a user in the *ccm_admin* role can create, modify, or delete a release definition.
- A user in the *ccm_admin* role can change the release delimiter.

- A build manager or a user in the *ccm_admin* role can rename a release if only the release definition and its associated process rules will be updated, and you must be in the *ccm_admin* role if other associated objects will be updated.

Options and arguments

`-accept` *dbid*

Specifies that updates are to be accepted from a specified database for a specified release or set of releases.

`-active`

Specifies that only active releases are listed. If not specified, both active and inactive releases are listed or modified.

`-baseline` *releasename*

Specifies the baseline release to be used when creating the new release.

`-component` *componentname*

Specifies a component name in the release - all of the release definitions matching that component name will be modified. The componentname can be the empty string "" meaning all releases for the null component name. This can be used in conjunction with the `controlling_database` option to receive control of all upgraded releases from some other database, or the `-list` option.

`-cdb|-controlling_database`

Sets DCM to either hand over to a specified database, or accept updates only from a specified database.

`-c|-create`

Creates a new release.

`-d|-delete releasename [-force]`

Deletes an existing release definition. If the release definition has any successor release definitions, these will be automatically history-collapsed. If there are any objects using the specified release name other than process rules for the release, the operation will fail if the `-force` option is not specified. If `-force` is specified and there are objects using the release name, a warning will be issued but the operation will be performed.

`-d|-delete -old releasename [-force]`

Deletes an release definition that has not been converted on upgrade. Use this command to remove releases that are not longer required. If there are any objects using the specified release name, the operation will fail if the `-force` option is not specified. If `-force` is specified and there are objects using the release name, a warning will be issued but the operation will be performed.

`-d|-delimiter`

Shows or changes the release delimiter used in release names.

`-description`

Specifies a one-line description of the release. If you need to enter a multi-line description, use the `description_file` option instead.

`-description_path`

Specifies a path to a file containing the description to be used.

`-force`

Suppresses confirmation messages and forces the rename or delete operation to be carried out. You can use this option only with the `-delete` and `-rename` options.

`-from` *releasename*

Specifies the release name to be copied when creating the new release.

`-groups`

Displays the groups (based on existing group security) that have permissions to modify or delete this definition.

`-handover`

Specifies that control of the database is to be handled by the selected database. This option can be used only when the release is locally controlled.

`-inactive`

Specifies that only inactive releases are listed. If not specified, both active and inactive releases are listed or modified.

`-included_releases`

Specifies one or many release names to be included in the release. This string supports multiple release names separated by a comma, and optionally, spaces. The comma is required; however, release names with leading or trailing spaces are not supported. Alternatively, you can use the `included_releases_file` option and enter data from a file.

`-included_releases_file`

Specifies a path to a file containing the releases to be included.

`-l|-list`

Lists release names.

`-local`

Specifies that control of the database is to be handled by the local database.

`-manager`

Specifies the product or component manager for the release name. The default on create is the user who is creating the release definition, and can be only a one-line string.

`-m|-modify`

Modifies an existing release.

`-nocheck`

Allows the `-rename` command to be started even though the user might not be in the *ccm_admin* role, the database might not be protected, or if other sessions are running on that database. If there are no objects to be renamed, then the command will attempt to rename the release. If there are any objects to be renamed, then the command will fail. The option delays the checks required for renaming objects until after the query has been performed to determine which objects are referenced in the release.

`-phase` *`phasename`*

Specifies a release phase value to indicate the phase of the release. The value must match one of the valid release phase values and is case sensitive. The default value is: `New, Requirements Definition, Function Definition, Implementation, Validation,` and `Released`.

`-preview`

Allows the `-rename` option to be run to show a summary of how many objects would be updated.

`-process` *`process_name`*

Allows you to specify a process for a release as it is being created. The release-specific process rules associated with the generic process rules for the specified process are associated with the new release. If any of the release-specific process rules do not exist, they will be created.

The `-process` option cannot be used with the `-purpose` or `-purposes_file` options.

`-purposes`

Specifies one or many purpose names to be included in the release. If one of the purposes specified has more than one generic process rule, then the process rule from the default process rule will be used. Each purpose name must be a valid purpose as defined in the project purpose table. This string supports multiple purpose names separated by a comma, and optionally, spaces. The comma is required; however, purpose names with leading or trailing spaces are not supported. Alternatively, you can use the `purposes_file` option and enter data from a file.

`-purposes_file`

Specifies a path to a file containing the purposes to be included in the release. Each purpose name must be entered on a new line.

`-releasename`

Specifies the release name. In most cases, `releasename` includes the component name, release delimiter, and component release value, such as `CM/6.4`. In the case where the `componentname` equals "None", the release name will be the component release value, such as `6.4`. This allows for backward compatibility.

The maximum allowed length of release component names is 64 characters.

`-r|-rename`

Renames an existing release. If a release definition exists for the specified *`oldreleasename`* it is renamed to the *`newreleasename`*. If an old release table entry exists for the *`oldreleasename`*, that entry is deleted and a release definition for the

newreleasename is created if one does not exist. All objects referencing the old release name will be updated to reference the new release name

If objects exist using the new release name, then this signifies a release merge. Such a release merge will fail with an error message if the `-force` option. For a release merge with `-force` specified, a warning will be issued but the operation will be permitted.

`-s|-show information` *release_name*

Shows all of the properties for the specified release.

`-s|-show property` *release_name*

Shows a specific property of the specified release. The following property keywords are supported:

```
active
allow_dcm_transfer
baseline
create_time
description
groups
included_releases
information
manager
modifiable_in
owner
parallel_check_out
parallel_check_in
phase
phase_log
purposes
```

Examples

- Delete the release definition for `Sweet 7.1`, regardless if any objects use the specified release name.

  ```
  ccm release -delete "Sweet/7.1" -force
  ```

- Create a new release `alphabets 2.0`, using the properties from `alphabets 1`.

  ```
  Windows:
  ccm release -create "alphabets/2.0" -from "alphabets/1.0" -
  description_file  c:\alphabets_2\features.doc\

  UNIX:
  ccm release -create "alphabets/2.0" -from "alphabets/1.0" -
  description_file  /usr/tom/alphabets_2/features
  ```

- Create a release for a new component (not based on an existing release) named `harmony 1.0`.

  ```
  ccm release -create "harmony/1.0" -desc "new product line to integrate
  X and Y" -manager "S Sweet" -active -noallow_dcm_transfer
  ```

- In `Widget release 5.2`, use the release information from the `5.1` and `5.0` releases when reconfiguring. Note that this operation is used mainly for object-based CM.

  ```
  ccm release -m "Widget/5.2" -included_releases "5.1,5.0"
  ```

- View the release names that have not been upgraded for the `Kit Ten` product.

  ```
  ccm release -list -inactive "Kit Ten"
  ```

- Show all active releases in Synergy

  ```
  ccm release -list -active "Synergy"
  ```

- View information about Synergy 7.1.

  ```
  ccm release -show information "Synergy/7.1"
  ```

- Edit the release information to show a new description, a new manager and that the release is in the implementation phase.

  ```
  ccm release  -modify -description "version a of release 1.0 without
  graphics capability" -manager sue -phase Implementation
  ```

- Hand over control of a locally-controlled release definition to a database whose ID is A1.

  ```
  ccm release -controlling_database -handover A1 -component releasename
  ```

resync command

Synopsis

```
ccm resync -f file_spec
```

Description and uses

The `resync` command updates the database with one or more objects from the work area. On UNIX, perform this operation only if you are using file copies in your project, or if you have deleted a link and replaced the link with a file copy.

Any user can execute this command.

Options and arguments

`-f file_spec`

Specifies the file(s) to be resynchronized.

Example

Resynchronize `clear.c` (update the database with the file from the work area).

```
ccm resync clear.c
```

Related topics

- reconcile command
- sync command

set command

Synopsis

```
ccm set [option [value]]
```

Description and uses

The `set` command enables you to set Rational Synergy options, display the values for options, and list options.

With no arguments, `set` lists Rational Synergy options and their values. If you specify an option name without a new value, the current value of the option is displayed. For example, to find out what role you are in, enter `ccm set role`, and the value are displayed.

The initial values for options are set in the initialization file.

Some of the options you can set include: `make_format`, `use_format`, `text_editor`, `text_viewer`, `role`, `verbosity`, and many more. See Default settings for a comprehensive list of options, and how and where to set them.

Options and arguments

option value

Specifies the name of the option to be set or displayed, and optionally, the value to set it to.

Examples

- Set your role to *developer*.

  ```
  ccm set role developer
  ```

- Display your current role.

  ```
  ccm set role
  developer
  ```

- Set the value of the `use_format` option.

  ```
  ccm set use_format "%displayname %status %owner %task %platform
%release"
  ```

- Show the value of `text_editor`.

  Windows:

  ```
  ccm set text_editor
  NOTEPAD.EXE %filename
  ```

  UNIX:

  ```
  ccm set text_editor
  vi %filename
  ```

Caveat

Some Rational Synergy options are set implicitly (`wa_type, sync_on_derive,` etc.) and cannot be modified with the `set` command.

Related topics

- unset command

show command

Synopsis

```
ccm show -p|-projects [-o|-owner owner]
        [-n|-name name] [-v|-version version]
        [-s|-state state] [-f|-format "format_string"]
        [-task task_number]
ccm show -t|-types
ccm show -mar|-migrate_auto_rules
```

Description and uses

The `show` command enables you to view the settings for certain attributes for projects, or to view all types in the database.

When used with the `-p` option, the `show` command lists projects that satisfy the criteria established with the other options.

When used with the `-t` option, the `show` command lists object types defined in the database.

When used with the `-mar` option, the `show` command lists the migrate rules that are auto-generated based on the type definitions.

Options and arguments

`-f|-format "format_string"`

Specifies a replacement string, which specifies the output format. Both text and Rational Synergy keywords are valid entries, for example, `"Name: %name, Type: %type"`. By default, the output format of this command shows `%displayname`, `%status`, `%owner`, `%type`, `%project`, `%instance`, and `%task`.

`"format_string"` uses keywords and literal text, such as:

```
Name: %displayname Owner: %owner
```

A keyword can be built-in (`%fullname`, `%displayname`, `%objectname`) or the name of any existing attribute such as `%modify_time` or `%status`.

See Built-In keywords for a list of keywords.

`-mar|-migrate_auto_rules`

Shows the migrate rules that are auto-generated based on the type definitions.

`-n|-name` *`name`*

Shows only objects with the name *`name`*.

`-o|-owner` *`owner`*

Shows only objects owned by *`owner`*.

`-p|-projects`

Shows projects in the database.

`-s|-state` *`state`*

Shows only objects in status *`state`*.

`-task` *`task_number`*

Shows only the objects associated with the specified task.

`-t|-types`

Shows the types in the database.

`-v|-version` *`version`*

Shows only objects with the version *`version`*.

Examples

- Show projects in the database that have the status *integrate* and owner *mike*.

```
ccm show -p -s integrate -o mary
1) projY-1   integrate mary project projY 1 2
2) projY-2   integrate mary project projY 1 7
3) projY-2.1 integrate mary project projY 1 8
```

- Show projects in the database that have the status *integrate*, owner *mary*, and are associated with task 8.

```
ccm show -p -s integrate -o mary -task 8
1) projY-2.1 integrate mary project projY 1 8
```

- Show the types defined in the database.

```
ccm show -t

ascii
binary
c++
csrc
dir
executable
incl
library
lsrc
makefile
project
relocatable_obj
shared_library
shsrc
symlink (UNIX)
ysrc
```

soad command

Synopsis

Preview / Create Object List

```
ccm soad -preview -scope "scope_name" [arg1 [arg2 [arg3 [arg4 [arg5 ]]]]]
                  [-so|-save_offline]
                  [-sort|-nosort]
                  [-f|-format "format"]
                  [-v|-verbose]
```

Delete Using Object List

```
ccm soad -delete [-pn|-package_name "package_name"]
                  [-path "path"]
                  [-v|-verbose]
```

Delete Using Scope

```
ccm soad -delete -scope "scope_name" [arg1 [arg2 [arg3 [arg4 [arg5 ]]]]]
                  [-so|-save_offline]
                  [-pn|-package_name "package_name"]
                  [-path "package_path"]
                  [-v|-verbose]
```

Prerequisites

To save objects offline, the current database must be initialized for DCM and a DCM license must be available.

Description and uses

The `soad` command enables you to preview and create an object list, to save objects offline, and to delete objects. The save operation creates a DCM transfer set with which you can restore the objects later.

The following are common types of data you might want to save offline and delete:

- Unwanted “Insulated Development” projects for specific developers
- Unwanted “Integration Testing” projects
- Old, static project hierarchies and the old files associated with them, where the hierarchies have been superseded by later released versions
- Unused old products

The following are the SOAD command roles and restrictions.

Scope Roles

Any user can view the available scopes. The roles defined for each scope determine whether you can use that scope to save offline and delete.

Rational Synergy Rules

In accordance with Rational Synergy rules, you can delete any *working*-state project or object owned by you. If you are working as a build manager, you can delete *prep* projects and objects. If you are in the *ccm_admin* role, you can save offline and delete objects in any non-working state or objects owned by other users. Only users in the *ccm_admin* role can save offline.

Work Areas

You can delete projects or objects if the work area is visible and writable. If you are a build manager or in the *ccm_admin* role, you can delete projects and objects if the work area is not visible or writable, but you might need to clean up the work area manually first.

Options and arguments

`-delete`

Deletes the database objects specified by the scope.

The scope for the deleted objects is determined in either of the following ways.

- from the accompanying `-scope` option
- from a previous preview of the object list (selection set)

The objects are saved before being deleted if either of the following is true:

- the `-so` option accompanies the command
- the `-so` option was used on a previous `ccm soad -preview` command

In either case, the deletion excludes (preserves in the database) any object that would be the last remaining version of that object name and instance.

`-f|-format "`*`format`*`"`

Enables you to change the default output format of the object list using Rational Synergy keywords. For the list of keywords, see Built-In keywords.

You can use this option only with the `-preview` option.

`-path "`*`package_path`*`"`

Specifies the path to the DCM package using the previously specified path.

If you have not saved a package previously, you must specify the path.

Note The path must be visible to the engine and writable by *ccm_root*.

`-pn|-package_name "`*`package_name`*`"`

Specifies the name of the DCM package to which the objects are saved. The default name is "`Save Offline and Delete saved on %date`."

Note You should include the `%date` keyword in the name if you define your own package name. Including the date keyword helps ensure that you can differentiate between packages created using the same scope.

`-preview`

Using the selected scope and optional arguments, creates an object list (selection set) that excludes (preserves in the database) any object that would be the last remaining version of that object name and instance, then lists the objects found.

You can use the object list as input to other Rational Synergy commands, including other `ccm soad` commands.

Note The preview results are not overwritten if you perform a query after a preview and before you use the object list for a `ccm soad -delete` command.

`-scope "`*`scope_name`*`" [`*`arg1`* `[`*`arg2`* `[`*`arg3`* `[`*`arg4`* `[`*`arg5`* `]]]]]`

Specifies the scope (modified "query") used to save offline or delete objects.

Arguments *`arg1`* through *`arg5`* are required only if appropriate for the specified scope. For example, the scope "`My working projects and products for a specified release`" requires that you specify a release value, which is *`arg1`*.

You must specify the arguments in the order used in the scope definition.

`-so|-save_offline`

When used with the `-preview` option, creates an object list that excludes (preserves in the database) any object that would be the last remaining version of that object name and instance.

When used with the `-delete` option, creates an object list that excludes (preserves in the database) any object that would be the last remaining version of that object name and instance, then saves the objects to a DCM package before deleting the objects.

`-sort|-nosort`

Sorts the preview output alphanumerically, or disables sorting. By default, the output is sorted.

`-v|-verbose`

Generates messages detailing why objects are being included in or excluded from the list.

Examples

- Save a project offline, then delete the objects in the project.

  1. Preview the list of objects to be deleted.

  ```
  ccm soad -pr -scope "scope_name" [arguments] -so
  ```

  Review the preview results to ensure that they are correct.

  > **Caution** Always preview the object list before saving the objects offline and deleting them. Choose a different scope, edit the scope, or create a new scope if the results are not correct.

  2. Save offline and delete the objects (using the object list).

  ```
  ccm soad -delete [-pn "package_name"] [-path "path"]
  ```

- Restore the objects in the specified SOAD transfer set.

  ```
  ccm dcm -rec -dir dir_path -ts "soad_scope_name"
  ```

  > **Note** You can restore the objects into a different database, or into a database for which the database ID has changed, by specifying the -dbid *database_id* option.

Related topics

- soad_scope command

soad_scope command

Synopsis

Create Scope

```
ccm soad_scope -c|-create "scope_name"
               [-roles role1 role2... roleN]
               [-parameters [label1 [|label2 [|label3 [|label4 [|
                 label5 ]]]]]]
               [-object object_spec | -query "query_expression"]
               [-expand|expansion_rules "expand_rules"]
               [-exclude|-exclusion_rules "exclude_rules"]
               [-exclude_query|-exclusion_query "query_expression"]
               [-pn|-package_name "package_name"]
```

Edit Scope

```
ccm soad_scope -m|-modify "scope_name"
               [-roles role1 role2... roleN]
               [-parameters [label1 [|label2 [|label3 [|label4 [|
               label5 ]]]]]]
               [-object object_spec | -query "query_expression"]
               [-expand|expansion_rules "expand_rules"]
               [-exclude|-exclusion_rules "exclude_rules"]
               [-exclude_query|-exclusion_query "query_expression"]
               [-pn|-package_name "package_name"]
```

List Scopes

```
ccm soad_scope -list
                [-s|-scope]
                [-expand|-expansion_rules]
                [-exclude|-exclusion_rules]}
```

Show Scope

```
ccm soad_scope -show "scope_name"
```

Delete Scope

```
ccm soad_scope -d|-delete "scope_name"
```

Prerequisites

None.

Description and uses

The `soad_scope` command edits, creates, and deletes scopes used to save offline and delete objects.

> **Caution** Before you create a new scope, you should start with an existing scope, preserve all exclusion rules, and validate the scope using test data.

You can edit, create, or delete a scope only when working in the *ccm_admin* role.

Options and arguments

`-d|-delete "`*`scope_name`*`"`

Deletes the specified scope.

`-exclude|-exclusion_rules '`*`rule1`*`' | '`*`rule2`*`' | ...'`*`ruleN`*`'`

Used with the `-list` option, lists the exclusion rules.

Used with the `-modify` or `-create` option, specifies one or more exclusion rules as follows. Exclusion rules remove related objects from the initial object list.

For example, if your query retrieves all objects for a specified release, with the release name as the first parameter (`release='%1'`), you can restrict the scope by adding exclusion rules to remove from the scope folders and tasks used by other projects; tasks used by other folders or associated with other objects; baselines used by other non-static projects; and objects that are part of other saved baselines.

For the text of the scope described above, see Release-based scope.

`-expand|-expansion_rules '`*`rule1`*`' | '`*`rule2`*`' | ...'`*`ruleN`*`'`

Used with the `-list` option, lists the expansion rules.

Used with the `-modify` or `-create` option, specifies one or more expansion rules. Expansion rules add related objects to the initial object list.

For example, if your query retrieves all objects for a specified release, with the release name as the first parameter (`release='%1'`), you can expand the scope by adding expansion rules to include the project's folder and tasks; the folders' tasks; and the tasks' objects.

For the text of the scope described above, see Release-based scope.

`-exclude_query|-exclusion_query "`*`query_expression`*`"`

Specifies a query used to remove objects from the scope.

For example, to exclude from the scope objects that have an attribute named `requirements`, specify the following query expression:

```
has_attr('requirements')
```

SOAD will add the following negated clause, wherever it evaluates an object name, query, or rule:

```
and not has_attr('requirements')
```

`-l|-list`

Shows all scopes.

The `-list` option requires that `-scope`, `-expand` or `-exclude` be specified.

`-m|-modify "`*`scope_name`*`"`

Edits the specified scope.

`-object `*`object_spec`*

Specifies the name of the object used for the initial object list (for example, `%1`). The resulting expanded string must be a valid 4-part object name.

For example, you can use the project object name, entered as the first parameter (`%1`), to set the initial object list to that project object name.

For the text of the scope described above, see Release-based scope.

`-parameters [`*`label1`*` [|`*`label2`*` [|`*`label3`*` [|`*`label4`*` [|`*`label5`*` ]]]]]`

Supplies labels for arguments for the `-object`, `-query`, and `-exclude_query` and definitions.

For example, define a scope such as the following for one parameter label, `Release Value`, for the query used in the "`All objects for specified release`" scope:

```
ccm soad_scope -create "All objects for specified release"
-parameters "Release Value" -query "release='%1'" other_options
```

Next, use the scope in the following ccm soad `-delete` command, where "2.3" is the release value:

```
ccm soad -delete -scope "All objects for specified release" 2.3
```

`-pn|-package_name "package_name"`

Specifies the name of the DCM package to which objects are saved for the scope. The package name can include keywords.

`-query "query_expression"`

Specifies the query expression that defines the initial object list.

For example, to make the initial object list include all the current user's projects and products for a specified release, specify the following query expression:

```
(cvtype='project' or is_product=TRUE) and owner='%user' and
status='working' and release='%1'
```

`-roles role1 role2... roleN`

Specifies one or more roles allowed to use the scope. By default, only users working in the *ccm_admin* role can use the scope.

`"scope_name"`

Specifies the scope for Save Offline and Delete.

Use only characters not restricted by the OS.

This name is also the scope's file name, including spaces and other characters, converted to a URL. For example, if you name the scope `This is my test scope`, the file name created is `This%20is%20my%20test%20scope.xml`.

`-show "scope_name"`

Shows all scopes, with the following details:

- Roles
- Parameter Labels
- Object
- Query
- Expansion Rules
- Exclusion Rules
- Exclusion Query
- Package Name

Examples

- View details of a specific scope.

  `ccm soad_scope -show "scope_name"`

- Create a new scope.
  1. List available scope names (to avoid using an existing name).

     ```
     ccm soad_scope -list
     ```

  2. Optionally, show expansion rule choices.

     ```
     ccm soad_scope -list -expansion_rules
     ```

  3. Optionally, show exclusion rule choices.

     ```
     ccm soad_scope -list -exclusion_rules
     ```

  4. Define the new scope.

     Caution Exclusion rules are usually required for scopes that remove an entire release.

     ```
     ccm soad_scope -create "scope_name"
     [-roles role1 role2... roleN]
     -object object_spec | -query "query_expression"
     [-expand_rules "expand_rules"]
     [-exclude_rules "exclude_rules"]
     [-exclude_query "query_expression"]
     [-parameters [label1 [|label2 [|label3 [|label4 [|label5 ]]]]]]
     [-package_name "package_name"]
     ```

  5. Verify the new scope.

     ```
     ccm soad_scope -show "scope_name"
     ```

- Edit a scope.
  1. List available scope names (to avoid using an existing name).

     ```
     ccm soad_scope -list
     ```

  2. Optionally, show expansion rule choices.

     ```
     ccm soad_scope -list -expand
     ```

  3. Optionally, show exclusion rule choices.

     ```
     ccm soad_scope -list -exclude
     ```

  4. Define the new scope.

     Caution Exclusion rules are usually required for scopes that remove an entire release.

     ```
     ccm soad_scope -modify "scope_name"
     [-roles role1 role2... roleN]
     -object object_spec | -query "query_expression"
     [-expansion_rules "expand_rules"]
     [-exclusion_rules "exclude_rules"]
     [-exclude_query "query_expression"]
     ```

```
[-parameters [label1 [|label2 [|label3 [|label4 [|label5 ]]]]]]
[-package_name "package_name"]
```

5. Verify the updated scope.

   ```
   ccm soad_scope -show "scope_name"
   ```

- Delete a scope.

  ```
  ccm soad_scope -delete "scope_name"
  ```

Related topics

- soad command

source command

Synopsis

```
ccm source filename
```

Description and uses

The `source` command executes the Rational Synergy commands found in the file *`filename`*. The Rational Synergy commands contained in *`filename`* should **not** include the "`ccm`" prefix. The `source` command is useful for running command sequences that you perform frequently.

Options and arguments

`filename`

Specifies the name of the file that contains the Rational Synergy commands.

Example

- Source the `ccm_product_cleanup` file, which selects floating products that are not being used in the database, then deletes them. (This script requires the user to be in the *ccm_admin* role.)

  ```
  ccm source ccm_product_cleanup
  ```

  The `ccm_product_cleanup` file contains the following commands:

  ```
  set role ccm_admin
  query -type executable "not is_bound()"
  collapse @
  query -type library "not is_bound()"
  collapse @
  set role developer
  ```

start command

Synopsis

```
ccm start [-nogui] [-q] [-d database_pathname] [-f filename]
          [-h engine_hostname] [-m] [-r initial_role]
          [-p project_spec] [-pw password][-u pathname]
          [-n user_name] (Windows only) [-home homedir]
          [-rc] (UNIX only)
```

Description and uses

The `start` command begins a Synergy Classic or CLI session by starting the engine and interface. After you enter the appropriate information in the Start dialog or on the command line, a progress bar shows the progress of the startup. After the session comes up, the Rational Synergy address (`CCM_ADDR`), which is a unique identifier for this interface session, is printed in your `ccm_ui.log`, in the Message View, and in your command window (Windows) or in the shell where you launched the session (UNIX).

If the CLI is is the only session that you are running, all Rational Synergy commands that you enter, from any command prompt (Windows) or shell (UNIX), from any machine on the local network that has access to your home directory, are executed by this Rational Synergy session.

If you set the `CCM_INI_FILE` environment variable to the path of a `ccm.ini` file (Windows), or `.ccm.ini` file (UNIX), that file is used for the startup and subsequent Rational Synergy commands.

If you run multiple Rational Synergy sessions, set the `CCM_ADDR` environment variable to specify which session will execute your Rational Synergy commands.

The type of connection used can affect performance. If the Rational Synergy server machine runs the engine process (the engine host) through a shared memory connection such as `ipcshm` protocol, you may have better performance compared to a remote connection (`soctcp` or `tlitcp` protocols).

> **Note** If you are running Synergy Classic or the CLI on a UNIX client and do not specify the engine host using the `-h` option on startup, the engine process is started on the local machine. For better performance, you can specify to run the engine process on the Rational Synergy server machine by using `ccm start -h server_name`

For Windows options and arguments, see Options and arguments (Windows).

For UNIX options and arguments, see Options and arguments (UNIX).

Options and arguments (Windows)

`-d` *`database_pathname`*

Specifies the absolute database path. The default is defined in your `ccm.ini` file.

`-f` *`filename`*

Specifies a different `ccm.ini` file, such as `ccm_test.ini`. You must use a full path name for the alternative initialization file (for example, `c:\ccm\new_inits\ccm.ini`). The path name **cannot** contain spaces.

`-h` *`engine_hostname`*

Specifies the machine on which the engine will run.

`-home` *`homedir`*

Specifies the path to the user's home directory.

`-m`

Permit multiple sessions. All Rational Synergy commands executed from a command prompt in which `CCM_ADDR` is set will use the interface process specified by that address. The address is displayed when the session is started.

For example: `set CCM_ADDR=murray:2775`

`murray:2775` is the `CCM_ADDR` value that would be displayed when you start your session.

Sessions started with this option require that `CCM_ADDR` be set before you can issue any commands.

`-n` *`user_name`*

Allows scripted session startup for a specified user.

`-nogui`

If `-nogui` is specified, Rational Synergy starts without GUI support. In this mode, only the CLI is available.

`-p` *`project_spec`*

Specifies the name and version of the startup focus project. If you do not specify a startup project, the focus project and project history stack from your previous session are loaded automatically by default.

`-pw` *`password`*

Specifies your password. This command is typically used to start sessions for scripting when you don't want the dialog brought up.

`-q`

Starts session in Quiet mode. When this option is used, these results occur:

- When the Synergy Classic session starts, the Startup screen is not displayed.
- The only output to stdout is the `CCM_ADDR` of the interface process.

If ESD is enabled, either the username and password must be entered, or the `.ccmrc` file needs to exist, in order to use this option, or you must configure your Synergy server for trusted hosts. For more information on trusted hosts, see the appropriate *Administration Guide*.

`-r` *`initial_role`*

Specifies the role that you are assigned at startup. The specified role must be a role permitted for you, or your session will not start.

`-u` *`pathname`*

Specifies the path name to which your database information is copied when you are running a remote client session. However, if your PC can access the database path, then you can leave this option out.

The default is `c:\temp\ccm`. You can change this location by using the `-u` option on the `start` command, or by setting `ui_database_dir` to the new path in the `[Options]` section of your `ccm.ini` file.

Examples

- Start Synergy Classic using the specified engine and database.

  ```
  ccm start -h cwi -d \\dbserver1\ccmdb\myproject
  ```

- Start a session in Quiet mode (without bringing up the Splash screen).

  ```
  ccm start -q -pw password -h engine_hostname -d database_path
  ```

For more information about the start command, see Caveats.

Options and arguments (UNIX)

`-d` *`database_pathname`*

Specifies the absolute database path. The default is defined in your `.ccm.ini` file.

`-f` *`filename`*

Specifies a different `.ccm.ini` file, such as `.ccm_test.ini`. You must use a full path name for the alternative initialization file (for example, `/users/sue/new_inits/.ccm.ini`). The path name **cannot** contain spaces.

`-h` *`engine_hostname`*

Specifies the machine on which the engine will run.

`-home` *`homedir`*

Specifies the path to the user's home directory.

`-m`

Permit multiple sessions. All Rational Synergy commands executed from a shell in which `CCM_ADDR` is set will use the interface process specified by that address. The address is displayed when the session is started.

For example: `export ccm ADDR=murray:2775:`

`murray:2775` is the `CCM_ADDR` value that would be displayed when you start your session.

Sessions started with this option require that `CCM_ADDR` be set before you can issue any commands.

`-nogui`

If `-nogui` is specified, Rational Synergy starts without GUI support. In this mode, only the CLI is available.

The output of Rational Synergy commands is displayed in your shell.

`-p` *`project_spec`*

Specifies the name and version of the startup focus project. If you do not specify a startup project, the focus project and project history stack from your previous session are loaded automatically by default.

`-pw` *`password`*

Specifies your password. This command is typically used to start sessions for scripting when you don't want the dialog brought up.

Sessions that connect across a firewall must specify a valid password using this option. If you don't specify a password, you will be prompted for a password before `ccm start` continues.

`-q`

Starts session in Quiet mode. When this option is used, the only output to stdout is the `CCM_ADDR` of the interface process.

`-r` *`initial_role`*

Specifies the role that you are assigned at startup. The specified role must be a role permitted for you, or your session will not start.

`-rc`

Specifies that you want to start a session as a remote client. The remote client mode will automatically be enabled for Rational Synergy sessions connected across a firewall.

A remote client session enables you to access a Rational Synergy database that is not visible from the interface—for example, a database that is not NFS-mounted on the interface host. (The database must, however, be visible to the engine.)

When you first start a remote client session, some of the database files are copied to a local directory called `/tmp/ccm/database_path`. You can change this location by using the `-u` option on the start command, or by setting `ui_database_dir` to the new path in the [Options] section of your `.ccm.ini` file.

You use file copies in your work area instead of files symbolically linked to database files when you are working in a remote client session.

`-u` *pathname*

Specifies the path name to which your database information is copied when you are running a remote client session. This option is used only with the `-rc` option.

The default is `/tmp/ccm`. You can change this location by using the `/-u` option on the `start` command, or by setting `ui_database_dir` to the new path in the `[Options]` section of your `.ccm.ini` file.

Examples

- Start Synergy Classic with the default Project View startup.

  ```
  ccm start
  ```

- Start Rational Synergy using the specified engine host and database.

  ```
  ccm start -h remoteHP -d /mnt/dev/ccmdb/myproject
  ```

- Create an alias or use a script with the `-q` option to start another session and set its address.

  ```
  alias ccmstart  export CCM_ADDR=`ccm start -m -q $*`
  ```

  OR

  ```
  #!/bin/sh
  export CCM_ADDR=`ccm start -m -q -nogui`
  ```

  Note: Use this method for `Rational Synergy` command scripts.

Caveats

If you start an additional Rational session and you plan to use the command line, a warning message is displayed. Set the `CCM_ADDR` variable for the new session to the address displayed by Rational Synergy start, for example:

```
set CCM_ADDR=prefect.cwi.com:1368
```

This causes your Rational Synergy commands to be executed by the new session rather than by the session you were already running.

When running as user *ccm_root*, always use the `-m` option and always set `CCM_ADDR` in the environment. This enables you to distinguish your *ccm_root* session from sessions where other users are running as *ccm_root*.

Environment Variables

`CCM_ADDR`

Files

`ccminit` (Windows) or `.ccminit` (UNIX) `-` (set of commands to be executed upon startup, for example, `alias r update`)

`ccm_ui.log` (user interface log file)

`ccm_eng.log` (engine log file)

`ccm.ini` (initialization file - Windows) or `.ccmi.ini` (initialization file - UNIX)

Related topics

- stop command

status command

Synopsis

```
ccm status
```

Description and uses

The `status` command displays information for your sessions. The information displayed includes the address of each interface process and the database being used. If you are in a work area, the name of the current project also is displayed.

Options and arguments

None

Example

- Obtain status on the current user.

```
ccm status

Rational Synergy sessions for user mary:

Graphical Interface @ toto:2531 (current session)
Database: /users/mb/devccmdb/test/db

Current project: 'rainbow platform_name 2.2'
```

- Obtain status on sessions running.

```
choochoo[121]: ccm status

Rational Synergy sessions for user npoulin:

Graphical Interface @ choochoo:34721:192.187.201.84
Database: /vol/dbserver.2/ccmdb/test_ccm51new

command Interface @ choochoo:34732:192.187.201.84
Database: /vol/dbserver.2/ccmdb/test_ccm51new

command Interface @ choochoo:34749:192.187.201.84
Database: /vol/dbserver.2/ccmdb/test_ccm51new
```

- Obtain status when no sessions are running.

```
choochoo[131]: ccm status
Rational Synergy sessions for user mary:

    No sessions found.
```

Related topics

- monitor command

stop command

Synopsis

```
ccm stop|quit
```

Description and uses

The `stop` command ends a Rational Synergy session.

Options and arguments

None

Example

- Stop the current Rational Synergy session.

```
ccm stop
```

Related topics

- start command

sync command

Synopsis

```
ccm sync [-r|-recurse] [-nr|-no_recurse]
         [-p|-project] project_spec [project_spec...]
```

Description and uses

The `sync` command creates or updates a work area for a project. The default directory in which all project work areas are created is `ccm_wa followed by the` database name in your home directory. Use the `sync` command to force a synchronization of the work area.

> **Note** Only a build manager or a user in the *ccm_admin* role can sync a non-writable project.

Your work area is created automatically when you create a project and when you check out a project using the check out commands. As you add new members to your project, your work area is updated automatically.

You will need to force a sync of your work area in the following cases:

- If you "clean out" (delete) any or all objects in your work area

  When you force a sync, only the necessary (controlled) objects from your database are written out to your work area.

- If the `work_area` command fails while changing your work area path

  When you change your work area path using the `work_area` command, the Work Area Properties dialog, or by moving the project, Rational Synergy will try to update your work area path to the new location. If another application is using the old work area path, the move will fail and you will need to synchronize your work area.

- If you change your work area type from one that uses local copies to one that uses symbolic links (or vice-versa)

  If you want to change your work area type, do the following:

  1. Reconcile the work area that you are currently using (either local copies or symbolic links).
  2. Delete the work area objects from the file system.
  3. Set your work area path and options.
  4. Start a new session using the client option of choice (either local copies or symbolic links).
  5. Re-create your work area by forcing a sync (execute the `sync` command).

If the work area already exists for a project and is the same type, you can use the reconcile command to update that work area (for example, after working disconnected from your database). The `reconcile` command is similar to the `sync` command, but has more conflict handling options.

Options and arguments

Note To stop a sync from the CLI, enter `CTRL+C` at any time.

Whenever you perform a sync from the CLI with a **Project View** open (Synergy Classic), the **Work Area Update Status** dialog is displayed.

To stop the sync, click the **Work Area Update Status** dialog's `Stop` button.

If you stop the sync, you will receive an error message stating that errors may occur in your work area. The errors will not occur until you try to use the work area; to avoid problems, perform a complete synchronization of the work area before you use it.

`-nr|-no_recurse`

Do not recurse the project hierarchy during the project sync. Synchronize only the specified project.

`-p|-project` *`project_spec`* `[`*`project_spec`*`...]`

Specifies the project that you want to synchronize.

`-r|-recurse`

Causes all objects in the project hierarchy to be sync'd along with the specified project. This is the default.

`-s|-static`

Updates an already-existing static work area with current data from the database (a static work area is a local copy of the work area for a static subproject). In addition, updates all static work areas in the hierarchy for which the `ccm sync` command was issued. This allows you to fully synchronize all static work areas in the hierarchy by using one command. If no static work area exists in the hierarchy, this option is ignored.

Example

- Synchronize the work area for `toolkit-mary` and its subprojects.

  ```
  ccm sync -recurse -project toolkit-mary
  ```

- Create a work area for the specified project.

  ```
  ccm sync -p ico_aug1-1
  ```

Defaults

You can set the following related options in your `ccm.ini file` (Windows) `or .ccm.ini` file (UNIX):

- save_to_wastebasket
- wastebasket
- wa_path_template
- sync_output

Related topics

- reconcile command
- resync command
- work_area command

task command

Synopsis

Assign a Task

```
ccm task -as|-assign task_specs -t|-to resolver -q|-quiet
```

Associate a Task with Objects, Existing Tasks, or Change Requests

```
ccm task -a|-associate|-relate task_spec
         {[-obj|-object file_spec [file_spec...]] | [-fixes task_spec] |
         [-cr|-change_request|-problem]  change_request_spec]}
```

Break a Relationship of a Task to Objects, Existing Tasks, or Change Requests

```
ccm task -d|-disassociate|-unrelate task_spec
         {[-obj|-object file_spec [file_spec...]] | [-fixes task_spec] |
         [-change_request|-problem]  change_request_spec]}
```

Complete a Task

```
ccm task -complete|-ci|-checkin task_spec|default
         [-c|-comment "string"]
         [-time|-time_actual task_duration] -y
```

Copy a Task

```
ccm task -cp|-copy -s|-synopsis "string"
         [-rel|-release release]
         [-p|-priority priority]
         [-r|-resolver resolver]
         [-sub|-subsystem subsystem]
         [-plat|-platform platform]
         [-time|-time_estimate time_estimate]
         [-date date_estimate] [-no_objects]
         [-register]
         [-description "description"]
         [-descriptionedit]
         [-descriptionfile file_path]
         [-def|-default] [-q|-quiet]
         task_specs
```

Create a Task

```
ccm task -cr|-create -s|-synopsis "string"
         [-p|-priority priority]
         [-plat|-platform platform]
         [-r|-resolver resolver]
         [-rel|-release release]
         [-sub|-subsystem subsystem]
         [-time|-time_estimate time_estimate]
         [-date date_estimate]
         [-description "description"]
         [-descriptionedit]
         [-descriptionfile file_path]
         [-def|-default] [-q|-quiet]
```

Fix a Task

```
ccm task -fix -s|-synopsis "string"
         [-rel|-release release]
         [-p|-priority priority]
         [-r|-resolver resolver]
         [-sub|-subsystem subsystem]
         [-plat|-platform platform]
         [-time|-time_estimate time_estimate]
         [-date date_estimate] [-register]
         [-exclude]
         [-description "description"]
         [-descriptionedit]
         [-descriptionfile file_path]
         [-def|-default] [-q|-quiet]
         task_specs
```

Modify Task

```
ccm task -mod|-modify
         {[-p|-priority priority]
         [-plat|-platform platform]
         [-r|-resolver resolver]
         [-rel|-release release]
         [-s|-synopsis "string"]
         [-sub|-subsystem subsystem]
         [-time|-time_estimate time_estimate]
         [-date|-date_estimate date_estimate]}
         [-description "description"]
         [-descriptionedit]
         [-descriptionfile file_path]
         task_specs -q|-quiet
```

Query for Tasks

```
ccm task -qu|-query query_spec
         [-f|-format "format_string"] [-ns|-no_sort] [-u]
         [-in_rel|-in_release] [old_project_spec] project_spec
         [-not_in_rel|-not_in_release] project_spec
```

Relate a Task to Objects, Tasks

```
ccm task -rel|-relate task_spec
         {[-obj|-object file_spec[ file_spec...]] | [-fix task_spec]}
```

Set or Clear the Current (Default) Task

```
ccm task -def|-default [task_spec|None]
```

Show Task Information

```
ccm task -sh|-show
         { i|info|information [-v|-verbose]| obj| objs| objects |
          -fix| -related [-all] | fixed_by | [-v|-verbose]}
         [-f|-format "format_string"] [-ns|-no_sort] [-u] task_specs
ccm task -sh|-show
         {p|priority |
         plat|platform |
         r|resolver |
         rel|release |
         s|synopsis |
         sub|subsystem |
         time|time_estimate |
         date|date_estimate |
         description |
         status_log
         cr|change_request|change_requests|prob|problem|problems}
         task_specs
```

Transition a Task to a Different State

```
ccm task -st|-state task_assigned | completed | excluded
         [-r|-resolver resolver]|
         [-description "description"]
         [-descriptionedit]
         [-descriptionfile file_path]
         task_specs
```

Unrelate a Task from Objects, Tasks

```
ccm task -unrelate task_spec
         {[-obj|-object file_spec[ file_spec...]] |
         [-fix task_spec]}
```

Description and uses

Use the `task` command to perform the following task-based operations:

- Assign a task
- Associate a task with objects, another task, or a change request
- Complete (check in) a task
- Copy a task
- Create a task
- Disassociate a task from objects, another task, or a change request
- Fix a task
- Modify a task
- Query for tasks
- Create (relate) or break (unrelate) relationships between a task and tasks, or objects
- Set or clear the current (default) task
- Show task information
- Transition a task to a different state

Options and arguments

`-a|-associate`

Associates the specified task with one or more tasks, objects, or change requests.

Use this command to fix a task with an existing task. To fix a task with a new task, see `-fix`.

The task with which you are associating objects or another task must be assigned and writable by you. When associating a change request, the change request must be writable by you and must be in a state that allows the association of tasks.

If the command is successful, the return value is `0`; otherwise, the return value is non-zero.

`-as|-assign`

Assign the one or more specified tasks to a resolver.

You must be working as an assigner or as PT administrator to use this option.

Use `-quiet` with this option to reduce the number of output messages displayed.

If the command is successful, the return value is `0`; otherwise, the return value is non-zero.

`-c|-comment "`*`string`*`"`

Enables you to add *`string`* as a comment to the task and associated objects you are checking in.

`-cr|-change_request|-problem`

Specifies the IDs of the change request on which you are performing an operation. For this argument's syntax, see Change request specification. Change request specifications can be separated by a comma or white space.

`-ci|-checkin`

Checks in either the current (default) task or the specified tasks. Note that "check in a task" is now called "complete a task" in Rational Synergy.

If you use the `-y` option and the task you are checking in is not associated with any objects, the confirmation message is suppressed.

If the command is successful, the return value is `0`; otherwise, the return value is non-zero.

`-complete`

Completes the current (default) task or the specified tasks. If you use the `-y` option and the completed task is not associated with any objects, the confirmation message is suppressed.

If the command is successful, the return value is `0`; otherwise, the return value is non-zero.

`-cp|-copy`

Enables you to copy a task. Copy a task when you need to apply a task that you fixed for the release to a different release. The copied task and the original task might have the same associated objects, different associated objects, or a combination.

The description of the copied task is not copied from the source task. Use `-description` or `-descriptionfile` to add a task description. Use `-descriptionedit` to bring up a text editor.

Use `-no_objects` if you do not want the objects associated with the source task to be associated with the copied task. Use `-register` to set the copied task's state to *registered*. Use `-resolver` to specify which user will complete the copied task.

By default, if you do not specify `-register`, the task is assigned to the same user as the source task or to the user specified by the `-resolver` option.

Use `-quiet` to reduce the number of output messages displayed.

`-cr|-create`

Enables you to create a task with the specified properties.

Use `-description` or `-descriptionfile` to append text to the existing task description. Use `-descriptionedit` to bring up a text editor displaying the existing task description. Use `-quiet` to reduce the number of output messages displayed.

If the command is successful, the return value is `0`; otherwise, the return value is non-zero.

`-date|-date_estimate` *`date_estimate`*

Enables you to estimate the date that the task will be be completed.

You can use this option only with the `-create, -copy, -fix,` or `-modify` option.

The `-show` option also can have a value of "`date_estimate`" (without the dash).

`-def|-default`

Enables you to set the current (default) task to the specified task number, to `None`, or to display the current task. Note that the default task is now called the current task in Rational Synergy.

When you use this option with the `-create` option, the new task is created, assigned to you, then set as the current task for the current Synergy Classic session. If you use the `-resolver` option to assign the task to another user, you will receive an error message and the task will not be created.

If the command is successful, the return value is `0`; otherwise, the return value is non-zero.

> **Note** Any current task you select must be in the *assigned* state and you must be the task's *resolver*.

If a current task is not set and you enter `-default`, you will receive a message stating that the default is not set. The return value from this command is 1 when the current task is not set.

`-description "`*`description`*`"`

Enables you to add a task description.

`-descriptionfile` *`file_path`*

Enables you to write a task description.

`-descriptionedit`

Brings up a text editor that displays the task description.

`-d|-disassociate`

Disassociates the specified task or change request from one or more tasks, or objects. Use this command to break the relationship with an existing task.

The task from which you are disassociating must be writable by you.

If the command is successful, the return value is `0`; otherwise, the return value is non-zero.

`-fix`

Creates a task and establishes a relationship between it and the task to be fixed; breaks a relationship between a task and the task it fixed.

Creating a fix task creates a relationship between the tasks. This enables Rational Synergy to detect when a project is using one task without the other. (This is called a conflict.) For information on conflicts, see conflicts command.

Use this command to fix a task by creating a relationship with a fix task.

To fix a task with an **existing** task, see `-relate` or `-a|-associate` option (both commands do the same thing; use `-relate` when possible). If the fix task contains a bug, create a new fix task to fix the first fix task.

Additionally, you can break a task relationship by using the `-unrelate` option. Break a fix relationship between two tasks when the fix relationship was set up incorrectly.

The following outlines task requirements for creating a fix relationship:

- Tasks related to each other can be from different databases.
- Tasks to be fixed must be in either the *completed* or *excluded* state.
- A fix task must be modifiable by the user establishing the relationship.
- A task can only fix one task.

The description of the fix task is not copied from the source task. Use `-description` or `-descriptionfile` to write a task description. Use `-descriptionedit` to bring up a text editor.

Use `-exclude` to set the state of the fix task to *excluded*. Use `-register` to set the state of the fix task to *registered*. Use `-resolver` to specify which user will complete the fix task.

By default, if you do not specify `-register`, the task is assigned to the same user as the source task or to the user specified by the `-resolver` option.

Use `-quiet` to reduce the number of output messages displayed.

See the `-modify`, `-associate`, or `-disassociate` commands to change values (that is, synopsis, resolver, platform, release, etc.).

`-fixes`

When used with `-associate|-relate`, establishes a relationship between two existing tasks; when used with `-disassociate|-unrelate`, deletes a relationship between two existing tasks.

`-f|-format "`*`format_string`*`"`

Specifies the command's output format. The default format depends on the other options you use with `-format` (that is, `-query` or `-show`) and those options' keyword arguments. See the options' descriptions for their default output formats.

The required string uses keywords and literal text, such as:

```
%displayname %owner
```

A keyword can be built-in (`%fullname`, `%displayname`, `%objectname`) or the name of any existing attribute such as `%modify_time` or `%status`.

See Built-In keywords for a list of keywords.

`-in_rel|-in_release [`*`old_project_spec`*`] `*`project_spec`*

Shows all tasks that are in the project hierarchy with *`project_spec`* as its root. This is determined by getting all tasks for all objects in the hierarchy, and for all their ancestors, subtracting the similar list of tasks for the hierarchy with *`old_project_spec`* as its root.

If *`old_project_spec`* is not specified, no tasks are subtracted. You should specify *`old_project_spec`* except for the first release of the product, when there is no baseline release.

`-m|-modify`

Enables you to change a task property using any combination of the following sub options:

`-s|-synopsis "`*`synopsis`*`"`
`-p|-priority `*`priority`*
`-r|-resolver `*`resolver`*
`-sub|-subsystem `*`subsystem`*

```
-plat|-platform platform
-time|-time_estimate time_estimate
-date|-date_estimate date_estimate
-rel|-release release
-q|-quiet
task_specs
```

Use `-description` or `-descriptionfile` to append text to the existing task description. Use `-descriptionedit` to bring up a text editor, displaying the existing task description. Use `-quiet` with this option to reduce the number of output messages displayed.

The `-modify` option accepts multiple sub options.

If the command is successful, the return value is `0`; otherwise, the return value is non-zero.

`-not_in_rel|-not_in_release` *`project_spec`*

Shows all tasks that are marked for a release but are not included in that release. This is determined by getting a list of completed tasks whose release matches the release of *`project_spec`* and subtracting all tasks for all objects in the hierarchy and their ancestors.

`-ns|-no_sort`

Do not sort the command's output.

`-obj|-object` *`file_spec`*`[ file_spec...]`

Specifies the name of the file or directory that is a member of the project.

If you specify more than one *`file_spec`*, you must leave at least one space between the file specifications.

You can use a selection set as an argument to this option if you are using the `-associate` or `-disassociate` option.

`-p|-priority` *`priority`*

Specifies the priority of the task you are creating. The priority can be `high`, `medium`, `low`, or `any`.

You can use this option only with the `-create` or `-modify` option.

`-plat|-platform` *`platform`*

Specifies the platform to which the change associated with the task applies. The platform choices are defined in the *`CCM_HOME`*`\etc\om_hosts.cf` file (Windows) or `$CCM_HOME/etc/om_hosts.cf` file (UNIX). If a task applies to multiple platforms, you should not set a platform value on the task.

You can use this option only with the `-create` or `-modify` option.

`-problem`

See `-cr|-change_request|-problem`.

`-qu|-query` *`query_spec`*

Displays the tasks that are found by the query performed using *`query_spec`*. Executing this command with the `-format` option populates a selection set with the tasks listed in the output.

You can control the format of the output by using the `-format` option. The default format is:

```
Task %nnnn: %task_synopsis
```

You can control the format of the output by using the `dbid` option to query a database other then your own.

```
Task %dbid#nnnn: %task_synopsis
```

where *`dbid`* is the Database ID, `#` is the DCM delimiter, and *`nnnn`* is the task number.

The syntax for *`query_spec`* is:

```
[-cus|-custom query_expression]
[-db|-database_id database_id]
[-plat|-platform platform]
[-rel|-release release]
[-sub|-subsystem subsystem]
[-ts|-scope|-task_scope task_scope]
```

where *`task_scope`* is one of the following:

```
user_defined
all_my_assigned
all_my_completed
all_my_tasks
all_completed
all_tasks
```

If you do not specify `-subsystem`, `-release`, `-platform`, or `-database_id`, they are assumed to have the value `Any`.

`query_expression` is the same as the *`query_expression`* for the query command command.

`-q|-quiet`

Reduces the number of output messages that are displayed.

`-relate`

Create a relationship between the specified task and one or more tasks, or objects.

Use this command to fix a task with an existing task. To fix a task with a new task, see `-fix`.

The task with which you are associating objects must be writable by you.

Use the `-fix` option to use an existing task to fix a task. Use the `-object` option to associate an object to the task.

If the command is successful, the return value is `0`; otherwise, the return value is non-zero.

`-rel|-release` *`release`*

Specifies the release to which the change associated with the task applies. You can view release choices by using the `ccm release` command.

You can use this option only with the `-create` or `-modify` option.

`-r|-resolver` *`resolver`*

Specifies which user is responsible for resolving the task. You can specify any one of the database users.

You must be working as the assigner or as the PT administrator to use this option.

You can use this option only with the `-create` or `-modify` option, and the task must be writable by you. If you use this option when you create or modify a task, the task is automatically assigned to the resolver that you specify.

`-s|-synopsis "`*`string`*`"`

Describes the task and is required. The string must be enclosed in quotes.

You can use this option only with the `-create` or `-modify` option.

`-sh|-show`

Shows the properties of the specified tasks. When you use this option with the `info`, `objs`, or `prob` keywords or their variants, a selection set is populated with the folders, objects, projects, and tasks listed in the output. Additionally, the `info` keyword will show tasks that are fixed by the current task and the tasks that the current task fixes.

Use the `-f` option to change the command's output format. Use `-u` to suppress automatic numbering of the output and `-ns` to suppress sorting.

Use `-show` with the `-fix` option to display the task that the current task fixes. Use `-show` with the `fixed_by` option to display the tasks that fix the current task. Use `-show` with the `related` option to display a list of completed tasks that fix or are fixed by the task. The tasks that fix and are fixed by tasks that are directly related are also displayed. To display tasks of all statuses, use the `-all` option with `-show related`.

You can use one of the following keywords if you are using the `-format` option:

```
i|info|information
obj|objs|objects
cr|change_request|change_requests|prob|problem|problems
```

Use `-show info` with the `-verbose` option to display all associated tasks and objects affected by the specified task. Use `-show info` with the `-description` option to display the `task_description` attribute of the tasks specified by *`task_spec`*. Use `-`

`show info` with the `-status_log` option to display the `status_log` attribute of the tasks specified by *`task_spec`*.

The default output format for `task -show information` is:

```
Task %displayname: %task_synopsis
```

where:

`%displayname` is `%name` if DCM is not enabled, and <*`database_ID`*><*`DCM_delimiter`*>`%name` if DCM is enabled.
`%task_synopsis` is a description of the task.

These lines are followed by additional information on some of the task's properties.

The default output format for `task -show objects` is:

```
%objectname %status %owner
```

where:

`%objectname` is the object's *`name-version:type:instance.`*
`%status` is the status of the object.
`%owner` is the owner of the object.

You also can show task properties using one of the following keywords if you are **not** using the `-format` option:

```
cr|change_request|change_requests|prob|problem|problems
p|priority
plat|platform
r|resolver
rel|release
s|synopsis
sub|subsystem
time|time_estimate
```

If the command is successful, the return value is `0`; otherwise, the return value is non-zero.

`-sub|-subsystem` *`subsystem`*

Specifies the subsystem to which the task belongs (for example, Any, GUI code, CLI code, or documentation). If the subsystem specification contains spaces, you must enclose it in quotes.

You can use this option only with the `-create` or `-modify` option.

`-st|-state task_assigned | completed | excluded`

Enables a developer to transition a task from the *completed* state to the *excluded* state and from the *excluded* state to the *completed* state if he is the resolver of the task. Enables a build manager, a PT administrator, or a user in the *ccm_admin* role to transition a task from the *completed* state to the *excluded* state and from the *excluded* state to the *completed* state.

You can use the `-resolver` *`resolver`* option only if you use the `-state task_assigned` option.

`-t|-to` *`resolver`*

Specifies the resolver to which you are assigning one or more tasks.

If you use this option with the `-assign` option, the task must be writable by you.

`task_spec`

Specifies the IDs of the tasks on which you are performing an operation. For this argument's syntax, see Task specification. Task specifications can be separated by a comma or white space.

> **Note** You can specify the name of a file containing a *`task_spec`* wherever you can specify *`task_spec`*.

`-time|-time_actual` *`task_duration`*

Enables you to specify the actual time required to complete the task.

You can use this option only with the `-checkin` option.

`-time|-time_estimate` *`time_estimate`*

Enables you to specify the estimated time required to complete the task.

You can use this option only with the `-create` or `-modify` option.

`-u`

Suppresses automatic numbering of this command's output ("un-numbered").

`-unrelate`

Breaks the relationship between the specified task and one or more tasks, or objects.

The task from which you are disassociating tasks, or objects must be writable by you.

Use the `-fix` option to break the relationship between an existing task and the task that fixed it.

If the command is successful, the return value is `0`; otherwise, the return value is non-zero.

`-y`

Suppresses a confirmation message when you are completing a task and the task is not associated with any objects.

Related topics

- task examples

task examples

View examples for the following operations:

- Associate a Task
- Assign a Task
- Complete (Check In) a Task
- Copy a Task
- Create a Task
- Disassociate a Task from Objects
- Disassociate a Task from a Change Request
- Fix a Task
- Modify Task
- Query for Tasks
- Set or Clear the Current (Default) Task
- Show Task Information
- Transition a Task

Associate a Task

- Associate task 17 with the object `MAIN.C-3:csrc:1`.

```
ccm task -a 17 -obj MAIN.C-3:csrc:1
```

- Associate task 54 with change request D#1231.

```
ccm task -associate 54 -change_request D#1231
```

Assign a Task

- Assign tasks 54, 60-63, and 74 to user *lseyer*.

```
ccm task -as 54,60-63,74 -to lseyer

Assigned task 54
Assigned task 60
Assigned task 61
Assigned task 62
Assigned task 63
Assigned task 74
```

Complete (Check In) a Task

- Check in all of the objects associated with task 40.

```
ccm task -checkin 40 -comment "The problem is fixed."

Object version is already associated with task.
Archiving CALC.H-4:source
Checked in 'CALC.H-4' to 'integrate'
Object version is already in the 'integrate' state: 'CALC.C-6'
Object version is already associated with task.
Archiving MSGS.H-5:source
Checked in 'MSGS.H-5' to 'integrate'
Summary:
1  skipped
2  succeeded
0  failed
Task '40' checked in.
```

- Complete the current task (check in the default task).

```
ccm task -ci default
```

Copy a Task

- Copy task 40, but give it a new synopsis, release, resolver, and description, and specify not to copy over the objects associated with it.

```
ccm task -copy 40 -synopsis "Fix GUI color problem" -release 2.0 -
resolver bob -no_objects -description "check RGB module"

Task hawaii#50 created.
```

Create a Task

- Create a new task with the synopsis (name) `Entanglement methods`.

```
ccm task -create -synopsis "Entanglement methods"

Task 44 created.
```

Disassociate a Task from Objects

- Disassociate task 35 from object version `MAIN.C-3:csrc:1`.

```
ccm task -d 34 -obj MAIN.C-3:csrc:1

Disassociated object version from task 34: MAIN.C-3:csrc:1
```

Disassociate a Task from a Change Request

- Disassociate task 10668 from change request `6569`.

```
ccm task -d 10668 -change_request 6569
```

Fix a Task

- Create a relationship between the fix task (19) and the task to be fixed (4).

```
ccm task -relate 19 -fixes 4
```

- Break a relationship between the fix task (25) and the task it fixed (12).

```
ccm task -unrelate 25 -fixes 12
```

Modify Task

- Change task 68's release to 4.1.

```
ccm task -modify -release 4.1 68

Changed release of task 68
```

Query for Tasks

- Query for the tasks that have a release value set to 3.0. Format the output so that it shows only the task synopsis.

```
ccm task -qu -rel 3.0 -f "%priority %task_synopsis"

1) high  Correct formatting of calculating number
2) high  Redesign gui for file open dialog
3) high  Performance improvement for file close
4) low   Enhance message text
```

Set or Clear the Current (Default) Task

- Show the current (default) task.

```
ccm task -default

The current task is not set.
```

- Set the current (default) task.

```
ccm task -default 26

The current task is set to:
26:  Close box no longer active
```

- Clear the current (default) task.

```
ccm task -default None
```

```
The current task has been cleared.
```

Show Task Information

- Show information on task 31.

```
ccm task -show info 31

Task:    31
 Synopsis:   Wrong window receives message
 State:      completed

 Resolver:        john
 Release:         3.1
 Priority:        high
 Subsystem:       <Uninitialized>
 Platform:        <Uninitialized>
 Database Id:     M

Task Description:
   The wrong window receives the event message when users abort an
operation. Currently, the Show window receives the abort message. The Home
window should receive this message.

Status Log:
Mon Aug 16 15:57:09 1999: Status set to 'registered' by mary in role
assigner
Mon Aug 16 15:57:14 1999: Status set to 'task_assigned' by mary in role
assigner
Tue Aug 17 11:16:55 1999: Status set to 'completed' by bill in role
developer
```

- Show formatted information on tasks 30 - 33.

```
ccm task -show info 30-34 -format "%priority %30-33 %task_synopsis" -
ns

1) high 33  Date field not validated on Inventory Form
2) high 41  Wrong window receives message
3) high 22 Saving a file takes forever
4) low 39  Button icons are rather obscure
5) low 4  OK button not default
```

- Show the change requests associated with task 68.

```
ccm task -show change_request 68

1) Change request 5
2) Change request 6
```

- Show the objects associated with tasks 4 and 5.

```
ccm task -show objects 4,5

1) MAIN.C-2:csrc:1  integrate  john
2) MAIN.H-4:incl:1  integrate  john
3) UTIL.C-7:csrc:1  integrate  john
4) MSGS.H-9:incl:1  integrate   john
```

Transition a Task

- Transition a task from completed to excluded.

```
ccm task -state excluded 94

Changed state of task KJG461#94 to excluded
```

- Transition a task from excluded to completed.

```
ccm task -state completed 94

Changed state of task KJG461#94 to completed
```

type command

The `type` command is an alias for the cat command.

typedef command

Synopsis

```
ccm typedef type_name -d|-description type_description
            -s|-super_type super_type_name
            [-f|-file_extension file_extensions]
            [-fw file_extensions] [-fu file_extensions]
            [-mm match_regular_expressions]
            [-mmw match_regular_expressions]
            [-mmu match_regular_expressions]
            [-mi TRUE|FALSE]
            [-miw TRUE|FALSE] [-miu TRUE|FALSE]

ccm typedef -i|-import type_name [-image] -dir from_path [-force]
ccm typedef -e|-export type_name -dir to_path [-force]
            [-ef|-export_format export_format]
```

Description and uses

Use the `typedef` command to add new types and update existing types in the current database. If you specify migrate rules for the type, these are stored as part of the type definition and are used for any automatically generated rules. When you create a new type, Rational Synergy automatically creates and stores its corresponding migration rules.

When you use the `-i` option, this command imports the specified type from the file system into the database. It must be a type that was previously exported from a Rational Synergy database using the `-e` option.

The `-e` option exports the specified type from the database into the file system. The directory to which you export the type must exist already, and must be writable by *ccm_root*.

You must be working as a type developer to use the `typedef` command.

Options and arguments

`-d|-description` *type_description*

Describes the type being added. If the description contains spaces, enclose the description in quotes.

`-dir` *from_path*|*to_path*

Specifies the full path to the directory from which the type is imported, or to which the type is exported. If the path contains spaces, enclose it in quotes.

`-e|-export` *type_name*

Exports the specified type from the database into the file system.

If you do not specify the -force option and the path is already populated, the path is not updated. If you specify the -force option, the exported representation of the type is overwritten.

-ef|-export_format *export_format*

Specifies that the export_format is either XML or CCM45SP2. The default is XML. If the format is XML, the type definition is represented as a single *XML* file named type.xml. This file contains the definition of the cvtype object and any attype object.

-f|-file_extension *file_extensions*

Specifies the default file extensions for the type. Use -f "" for no file extension.

-force

Forces an existing type to be overwritten with the new type, and adds any new attributes. You can use this option only with the -import and -export options.

Note Use the -image option, instead, if you want to remove attributes from an existing type because those attributes are not present in the new type.

-fu file_extensions

Specifies the type's default file extensions for use with a UNIX client. This is a list of one or more file suffixes separated by at least once space. Begin each suffix with a period ('.').

-fw file_extensions

Specifies the type's default file extensions for use with a Windows client. This is a list of one or more file suffixes separated by at least once space. Begin each suffix with a period ('.').

-g

Brings up the appropriate dialog.

-i|-import *type_name*

Imports the specified type from the file system into the database.

Note To import into an existing type, the type must be visible to the engine. You also must use either the -force option or the -image option to overwrite the existing type.

Caution If you import using the `-force` or `-image` option, the changes apply to all objects of the specified type in the database.

`-image`

Replaces all attributes, for all objects of the specified type, with the attributes in the import directory. New attributes are added, changed properties are replaced, and deleted attributes are removed from the database. You can use this option only with the `-import` option.

`-mi TRUE|FALSE`

Specifies, for the current client, whether objects of the specified type should be ignored on Migrate. A value of `TRUE` means ignore, a value of `FALSE` (default) means don't ignore.

`-miu TRUE|FALSE`

Specifies, for the UNIX client, whether objects of the specified type should be ignored on Migrate. A value of `TRUE` means ignore, a value of `FALSE` (default) means don't ignore.

`-miw TRUE|FALSE`

Specifies, for the Windows client, whether objects of the specified type should be ignored on Migrate. A value of `TRUE` means ignore, a value of `FALSE` (default) means don't ignore.

`-mm match_regular_expressions`

Specifies, for the current client, any regular expressions that should be used for file matching in the migrate rules. The value should be a list of one or more regular expressions separated by one or more spaces. See Migration rules for further information on regular expressions.

`-mmu match_regular_expressions`

Specifies, for the UNIX client, any regular expressions that should be used for file matching in the migrate rules. The value should be a list of one or more regular expressions separated by one or more spaces. See Migration rules for further information on regular expressions.

`-mmw match_regular_expressions`

Specifies, for the Windows client, any regular expressions that should be used for file matching in the migrate rules. The value should be a list of one or more regular expressions separated by one or more spaces. See Migration rules for further information on regular expressions.

`-s|-super_type` *`super_type_name`*

Specifies the super type of the type being added – that is, the type from which it should inherit its characteristics. Use the `show` command to list the types defined in the database.

Examples

- Create an object type for HTML files, with its characteristics inherited from the `ascii` super type, and with a `.html` extension for the current type of Rational client.

  ```
  ccm typedef html -d "Hypertext Markup Language" -s ascii -f ".html"
  ```

- Create an object type for JPEG files, with its characteristics inherited from the binary type, and with .jpeg, .jpg, and .jpe suffixes for both Windows and UNIX clients.

  ```
  ccm typedef jpeg -d "JPEG Image" -s binary -fw ".jpeg .jpg .jpe" -fu
  ".jpeg .jpg .jpe"
  ```

- Create an object type for listing files, with: 1) Its characteristics inherited from the ascii type, 2) “.lst” suffix on Windows, 3) “.lis” suffix on UNIX, and 4) Ignore by default on Migrate on UNIX clients:

  ```
  ccm typedef list -d "Listing file" -s ascii -fw ".lst” -fu ".lis" -miu
  TRUE
  ```

- Create an object type for MS word documents, with: 1) Its characteristics inherited from the binary type, and 2) A migrate match rule that recognizes .doc and .dot suffixes on Windows clients:

  ```
  ccm typedef msword -d "MS Word" -s binary -mmw ".*[Dd][Oo][CcTt]"
  ```

- Windows:
  Export the `pascal` type from the current database to the `c:\ccm\exported types` directory.

  ```
  ccm typedef -export pascal -dir "c:\ccm\exported types"
  ```

  UNIX:
  Export the `pascal` type from the current database to the `/mnt/ccm/exported types` directory.

  ```
  ccm typedef -export pascal -dir "/mnt/ccm/exported types"
  ```

- Windows:
  Import the `fmdoc` type into the current database from the `c:\ccm\types_to_import` directory.

  ```
  ccm typedef -import fmdoc -dir c:\ccm\types_to_import
  ```

  UNIX:
  Import the `fmdoc` type into the current database from the `/mnt/ccm/types_to_import` directory.

  ```
  ccm typedef -import fmdoc -dir /mnt/ccm/types_to_import
  ```

Caveat

You cannot change built-in types using the `typedef` command.

unalias command

Synopsis

```
ccm unalias alias_name
```

Description and uses

The `unalias` command removes a defined alias.

Using the `unalias` command removes an alias for the current session only.

Options and arguments

`alias_name`

Specifies the name of the alias you want to remove.

Example

Unalias the `getf` command.

```
ccm unalias getf
```

Related topics

- alias command

unrelate command

Synopsis

```
ccm unrelate -n|-name rel_name -f|-from file_spec1
             -t|-to file_spec2
```

Description and uses

The `unrelate` command deletes a relationship, *`rel_name`*, between *`file_spec1`* and *`file_spec2`*.

More relationships are predefined in Rational Synergy. See Relationships for a table of these relationships. However, you can define new relationships using the `relate` command.

To delete the relationship between two objects, you must include all three object specifications (defined below).

Options and arguments

`-f|-from` *`file_spec1`*

Deletes the relationship(s) originating with *`file_spec1`*.

`-n|-name` *`rel_name`*

Deletes the relationship(s) designated by *`rel_name`*.

`-t|-to` *`file_spec2`*

Deletes the relationship(s) to *`file_spec2`*.

Example

Delete the **successor** relationship from `clear.c-2` to `clear.c-1`:

```
ccm unrelate -n successor -f clear.c-1:csrc:1 -t clear.c-2:csrc:1
```

Related topics

- relate command

undo_reconfigure command

The `unreconf|undo_reconfigure` command is an alias for the undo_update command.

undo_update command

Synopsis

```
ccm unupd|undo_update|unreconf|undo_reconfigure
                        [-v|-verbose] [-r|recurse] file_spec
ccm unupd|undo_update|unreconf|undo_reconfigure
                        [-v|-verbose] [-r|-recurse]
                        -p|-project project_spec
ccm unupd|undo_update|unreconf|undo_reconfigure
                        [-v|-verbose] [-r|-recurse]
                        -pg|-project_grouping project_grouping_spec
```

Description and uses

The `undo_update` command reverses the update (reconfigure) operation for a specified directory or project object.

By default and for performance purposes, the `undo_update` command does not provide parallel version notification when it encounters parallel object versions. You can enable parallel version notification by setting the reconfigure_parallel_check user option to `TRUE` in your initialization file.

The undo update process stops if an individual operation within the undo fails. For example, if the current version of an object has a work area conflict, the process stops and the new version is not automatically used. This is done to protect the data in the user's work area.

The default setting to stop the undo update can be changed by modifying your initialization file. Some users may want to continue with the undo update process, even though an individual failure has occurred. You can set update to continue by setting the reconf_stop_on_fail option to `False`.

The `undo_update` command can be used to reverse the last undo update operation. In other words, if two or more undo updates are performed, only the last one will be reversed.

Options and arguments

`file_spec`

Specifies the directory where the update will be reversed.

`-p|-project` *`project_spec`*

Specifies the project where the update will be reversed.

`-pg|-project_grouping` *`project_grouping_spec`*

Specifies that all projects in the project grouping have their updates reversed. No changes are made to the project grouping's baseline and tasks.

`-r|-recurse`

Specifies that subprojects should be included.

`-v|-verbose`

Displays detailed on the undo update messages.

Example

- Undo the update on the `proj1-1` project.

  ```
  ccm unupd -p proj1-1
  ```

- Undo an update on a project named `toolkit-mary`, which has subprojects to be reversed.

  ```
  ccm undo_update -recurse -project toolkit-mary
  ```

Related topics

- update command

unset command

Synopsis

`ccm unset` *option*

Description and uses

The `unset` command removes any settings to the value of an option.

Options and arguments

option

Specifies the name of the option for which the value is being unset.

Example

Unset the `proj_log` option.

```
ccm unset proj_log
```

Caveat

Some Rational Synergy variables are set implicitly and cannot be unset with the `unset` command.

Related topics

- set command

unuse command

Synopsis

```
ccm unuse [-t|-task task_number]
          [-d|-delete] [-r|-replace]
          file_spec [file_spec...]
ccm unuse [-t|-task task_number]
          [-d|-delete] [-r|-replace]
          -p|-project project_spec [project_spec...]
ccm unuse -delete -force file_spec [file_spec...]
```

Description and uses

Removes an existing file, directory, root directory, or project from the current project or directory. The directory must be checked out to remove members from it; however, if you try to remove an object from a non-modifiable directory, Rational Synergy checks out the directory automatically (unless you specify the `-r` option). You must check **in** the directory to make the changes in the directory available to other users. Note that unuse is now called cut Rational Synergy.

The root directory **can** be the target of this command, but only when specified with the `-d` and `-r` options. If you want to use a different version of the root directory, use the `use` command. You cannot cut the root directory without replacing it because a project must always have a root directory.

> **Note** When you cut an object in a non-modifiable directory, a new directory version is checked out automatically unless you replace the object with a different version.
>
> If you are in a shared project and your current directory is non-modifiable, the directory is checked out and associated automatically with the current (or specified) task and is checked in to the *integrate* state. You can disable the automatic check-in feature by setting `shared_project_directory_checkin` to `FALSE` in your initialization file. (See shared_project_directory_checkin.)

If you want to delete a project, see the delete command (`ccm delete -p` *`project_name-version`*).

You do not need to be in a work area to use this command as long as you use the project reference form:

Windows: *`relative_path\object_name@project_name-project_version`*

UNIX: *`relative_path/object_name@project_name-project_version`*

The following is an example of the project reference form and how to use it to delete the root directory, `ico/hi_world.c@final-1`:

```
ccm unuse -d -r final@final-1
```

Options and arguments

`-d|-delete`

Remove the object from the directory, then delete it from the database.

You cannot cut and delete a project. When you perform this operation on a project, the project is cut but not deleted. Use the `delete` command to delete a project.

`file_spec`

Specifies the object or objects to unuse.

`-force`

The -force option suppresses confirmation messages and forces the delete operation to be carried out.

`-g`

Brings up the appropriate dialog.

`-p|-project` *`project_spec`*

Specifies the project or projects to cut.

If you are removing a subproject or subprojects, you must change to the directory that contains the subprojects, and you do not need to include the `-p` option.

`-r|-replace`

Replace the object in the directory with its predecessor. When this option is specified, the list of files in the directory remains unchanged; only the version of the specified object changes.

`-t|-task` *`task_number`*

Associates the newly checked-out directory with a task number, if an `unuse -delete` deleted an object from a read-only directory.

If the current (default) task is set and you do not specify a different task, the objects you are checking out are associated with the current task automatically.

Examples

- Cut the `sort.c` object from the current project.

  ```
  ccm unuse sort.c

  Member sort.c-1 removed from project ico-1
  ```

- Remove the `ico_jan5` and `ico_jan6` subprojects from the `ico_jan4-1` top-level project.

  ```
  ccm unuse ico_jan5 ico_jan6

  Member ico_jan5-1 removed from project ico_jan4-1
  Member ico_jan6-1 removed from project ico_jan4-1
  ```

- Delete and replace an object from your project.

  ```
  ccm unuse -delete -replace object_version
  ```

  You will receive a message telling you the *`object_version`* that was removed and which version replaced it.

Related topics

- use command

update command

Synopsis

```
ccm u|update|update_members|reconf|reconfigure
                        [-v|-verbose] [-r|-recurse]
                        [-rs|-replace_subprojects | -ks|-keep_subprojects]
                        file_spec
ccm u|update|update_members|reconf|reconfigure
                        [-v|-verbose] [-r|-recurse]
                        [-rs|-replace_subprojects | -ks|-keep_subprojects]
                        -p|-project project_spec
ccm u|update|update_members|reconf|reconfigure
                        [-v|-verbose] [-r|-recurse]
                        [-rs|-replace_subprojects | -ks|-keep_subprojects]
                        [-pg|-project_grouping]
                        project_grouping_spec
```

Description and uses

The `update` command updates the specified directory, project object, or project grouping. It uses the baseline and tasks of project groupings to find the appropriate candidates and selection rules to select new versions of the members, if appropriate. You can also specify a project grouping to be updated. Note that the reconfigure or update members operation is now called update in Rational Synergy.

The update process stops if an individual operation within the update fails. For example, if the current version of an object has a work area conflict, the process stops and the new version is not automatically used. This is done to protect the data in the user's work area.

The default setting to stop the update can be changed by modifying your initialization file. Some users may want to continue with the update process, even though an individual failure has occurred. You can set update to continue by setting the reconf_stop_on_fail option to `False`.

The use of project groupings enables you to more easily perform a multiphase build, where the set of projects in a project grouping is not updated all at once. In such a case, all the projects are updated using the same baseline and tasks. A developer or build manager can update additional projects in the same project grouping, using the same baseline and tasks, without having that baseline and those tasks refreshed. Thus, it is necessary to calculate and save this baseline and tasks, and to specify that subsequent project updates use this saved baseline and tasks.

You may need to change the update properties of a project. The update properties determine whether a project is updated using tasks and a baseline or object status, and enable you to set parameters that control which objects are selected. Here are some of the properties you may need to change, and the commands to be used to change them:

- Change the project's release or purpose with the `ccm attr` command.

- Switch between process-rule-based and manual updates with the `ccm update_properties` command).
- Switch between object status-based and task-based updates with the `ccm update_properties` command
- Add and remove tasks for process-rule-based projects, with the `ccm project_grouping` command.
- Set the baseline for custom development purpose process-rule-based projects, with the `ccm project_grouping` command.
- Add and remove tasks, and set the baseline, for manual projects, with the `ccm update_properties` command.

You can also change update properties using the Rational Synergy GUI.

By default and for performance purposes, the update command does not provide parallel version notification when it encounters parallel object versions. You can enable parallel version notification by setting the reconfigure_parallel_check user option.

Options and arguments

file_spec

Specifies the directory to be updated.

-ks|-keep_subprojects

Specifies that subprojects are to be kept in place. This is the default for the current session, unless the replace_subproj has been changed so that subprojects are replaced. If an update command is issued without the `-ks` or `-rs` option, the existing default is used.

-p|-project *project_spec*

Specifies the project to be updated.

-pg|-project_grouping

Specifies that the entire project grouping should be updated. The update properties are refreshed according to the settings on the project grouping. For additional information on project groupings, see the project_grouping command.

project_gouping_spec

Specifies the project grouping to be updated.

This can be entered as the four-part name, or as the default displayname for the grouping, such as `All CM/6.4 Integrations Testing Projects` or `Linda's CM/6.4 Collaborative Development Projects`. For more information, see Project grouping specification.

`-r|-recurse`

Specifies that subprojects should be updated.

`-rs|-replace_subprojects`

Specifies that subprojects can be replaced with new subprojects. Subprojects are replaced only if the other versions of those subprojects are selected by the selection rules. If an update command is issued without the `-ks` or `-rs` option, the existing default is used.

`-v|-verbose`

Specifies that detailed update messages are to be displayed.

Examples

- Update a project named `proj1-1` and replace its subprojects.

  ```
  ccm update -rs -p proj1-1
  ```

- Update all projects in the grouping named `All Fox/2.01 Integration Testing Projects`.

  ```
  ccm update -pg "All Fox/2.01 Integration Testing Projects"
  ```

Related topics

- undo_update command
- project_grouping command

update_properties command

Synopses

Compare Update Properties

```
ccm up|update_prop|update_properties|rp|reconf_prop|reconfigure_properties -
comp|-compare
       project_spec1 project_spec2
```

Add Tasks and/or Folders

```
ccm up|update_prop|update_properties|rp|reconf_prop|reconfigure_properties -
a|-ad|-add
       [-t|-task|-tasks task_specs] [-y] [-related]
       [-fol|-folder|-folders folder_specs]
       [-r|-recurse] [-q|-quiet] project_specs
```

Remove Tasks and/or Folders

```
ccm up|update_prop|update_properties|rp|reconf_prop|reconfigure_properties -
rem|-remove
       [-t|-task|-tasks task_specs] [-related]
       [-fol|-folder|-folders folder_specs]
       [-r|-recurse] [-q|-quiet] project_specs
```

Show Baseline, Tasks, Folders, and/or Objects

```
ccm up|update_prop|update_properties|rp|reconf_prop|reconfigure_properties -
sh|-show
       {at|all_t|all_tasks|
       b|bl| baseline_project
       fol|folders|
       obj|objects|
       t|tasks|
       tf|t_and_f|tasks_and_folders}
       [-auto|-automatic|-no_auto|-no_automatic]
       [-f|-format "format_string"]
       [-ns|-no_sort] [-u] [-r|-recurse] [-v|-verbose]
       project_spec
```

Show Update Properties

```
ccm up|update_prop|update_properties|rp|reconf_prop|reconfigure_properties -
sh|-show
       i|info|information
       [-v|-verbose] [-r|-recurse] project_specs
```

Set the Update Method

```
ccm up|update_prop|update_properties|rp|reconf_prop|reconfigure_properties -
ru|-reconf_using
       {os|obj_stat|object_status | t|tasks | template | manual}
       [-r|-recurse] [-q|-quiet] project_specs
```

Set Baseline for Specific Project

```
ccm up|update_prop|update_properties|rp|reconf_prop|reconfigure_properties
       -mb|-modify_baseline_project {baseline_spec|base_project_version}
       [-r|-recurse] -q|-quiet project_specs
```

Refresh Update Properties

```
ccm up|update_prop|update_properties|rp|reconf_prop|reconfigure_properties -
refresh [-recurse]
       project_specs
```

Save Update Properties to Subprojects

```
ccm up|update_prop|update_properties|rp|reconf_prop|reconfigure_properties -
ats|-apply_to_subprojs
       [-no_baseline] [-no_tasks_and_folders] project_specs
```

Show Valid Baseline Projects for a Specific Project

```
ccm up|update_prop|update_properties|rp|reconf_prop|reconfigure_properties -
vb|-valid_baseline_projects
       [-f|-format "format_string"] [-ns|-no_sort] [-u] project_spec
```

Description and uses

The `update_properties` command displays and sets the update properties on one or more projects, and allows you to compare the update properties of two projects.

A project's update properties consist of:

- The baseline and tasks on the project's project grouping, if the project is set to update using a process rule, or
- The baseline and tasks on the project, if the project is not set to use a process rule.

If you do not specify a project, the command applies to the project that is associated with the work area in which you execute the command.

Use the `update_properties` command to perform the following operations:

- Compare update properties.
- Add tasks and/or folders to a project.
- Removes tasks and/or folders from a project.
- Show baseline, tasks, folders, and/or objects for one or more projects
- Show update properties for a project.
- Set a project's update method.
- Set the baseline for a specific project.
- Apply update properties to subprojects.
- Update the baseline, folder, and tasks.
- Show the valid baselines for a specified project.

Options and arguments

`-a|-add`

Adds the specified task or folder to the update properties of *`project_spec`*. You can use this option only on projects that are writable by you.

If you add a task that is in the *excluded* state to a folder, you will receive a confirmation message that shows the state of the task and asks for confirmation. You can turn off this option by using the `-y` option.

Use `-quiet` to reduce the number of output messages displayed. Use `-r` to add a specified task or folder recursively to the update properties.

Use `-related` with `-add` and `-tasks` to add tasks to a specified folder or project. If you add a task that is in the *excluded* state to a folder, you will receive a confirmation message that shows the state of the task and asks for confirmation. Additionally, if you add a completed fix task to a folder without adding the task that it fixed, you will receive a message asking if you want to add the related task(s). For best results, add the related task or tasks.

If the command is successful, the return value is `0`; otherwise, the return value is non-zero.

`-ats|-apply_to_subprojs`

Sets the scope of the update properties to all projects in the specified project's hierarchy. Alone, this argument can be used to save the specified project's update properties to its subprojects.

`-auto|-automatic`

Specifies that automatic tasks are to be displayed.

You must use one of the following commands with this option:

- `ccm up -show tasks`
- `ccm up -show all_tasks`
- `ccm up -show tasks_and_folders`

This option overrides the values that are set in the `ccm.ini` file (Windows) or `.ccm.ini` file (UNIX). The default for this option is `FALSE`; automatic tasks are not displayed by default.

`-comp|-compare` *`project_spec1 project_spec2`*

Compares two projects' update properties.

`-f|-format "`*`format_string`*`"`

Specifies the command's output format. The default format depends on the other options used with the command.

The format can contain a combination of text and keywords. Keywords are replaced by specific information about each object as it displays. For example, the keyword `%owner` is replaced with `sue` if an object owned by user *sue* is displayed.

The name of any existing attribute can be used as a keyword. In addition, a number of built-in keywords are defined, such as `%displayname` and `%task_number`. See Built-In keywords for a list.

`-fol|-folder|-folders` *`folder_spec`*

Specifies the IDs of the folders that you are adding, listing, or removing. For this argument's syntax, see Folder specification.

> **Note** You can specify the name of a file containing a *`folder_spec`* wherever you can specify *`folder_spec`*.

`-mb|-modify_baseline` *`base_project_spec|base_project_version`*

Sets the baseline to *`base_project_spec`* project or *`base_project_version`* project version for the project specified by *`project_spec`*.

Use `-quiet` with this option to reduce the number of output messages displayed.

You can use this option only on projects that are writable by you.

If the command is successful, the return value is `0`; otherwise, the return value is non-zero.

`-no_auto|-no_automatic`

Specifies automatic tasks not to display.

You must use one of the following commands with this option.

- `ccm up -show tasks`
- `ccm up -show all_tasks`
- `ccm up -show tasks_and_folders`

This option overrides the values set in the `ccm.ini` file (Windows) or `.ccm.ini` file (UNIX). The default for this option is `FALSE`; automatic tasks are not displayed by default.

`-no_baseline`

Specifies that the project hierarchy's baseline project is not applied to the subprojects. This option is used with the `-apply_to_subprojs` option.

`-no_tasks_and_folders`

Specifies that the project hierarchy's tasks and folders are not applied to the subprojects. This option is used with the `-apply_to_subprojs` option.

`-ns|-no_sort`

Do not sort the command's output.

`-quiet`

Reduces the number of output messages. This option is useful for scripting.

`project_specs`

Is more than one *`project_spec`*.

`-r|-recurse`

Applies the current command to subprojects in the project hierarchy as well as the specified project.

`-refresh`

Updates the baseline, folder, and tasks on a project to make them consistent with the process rule. Updating the folder includes performing a query.

The update applies to subprojects only if you use the `-recurse` option.

This option is valid only for projects that are updated using a process rule and for which the update properties are modifiable by the current user in his or her current role.

`-related`

Use `-related` with `-tasks` and `-add` or `-remove`. This allows you to either:

- Add all tasks that are related to the specified fix task to the specified folder or project,

OR

- Remove all tasks that are related to the specified fix task from the specified folder or project.

`-rem|-remove`

Removes the specified task or folder from the update properties of the specified project. You can use this option only on projects that are writable by you.

Use `-quiet` with this option to reduce the number of output messages displayed. Use `-recurse` to recursively remove a specified task from the update properties.

If the command is successful, the return value is `0`; otherwise, the return value is non-zero.

`-ru|-reconf_using`

Specifies the method that is to be used to update the specified project(s). The options are task-based (`t|tasks|template|manual`) or object-status-based (`os|obj_stat|object_status`).

Use `-q` with this option to reduce the number of output messages displayed. Use `-r` to change the update method for a project recursively.

`-sh|-show`

Shows the specified type of object that is included in the *`project_spec`* project's update properties. A selection set is populated with the folders, objects, projects, and tasks listed in the output.

You must use one of the following arguments with the `-show` option.

`at|all_t|all_tasks -auto|-automatic | -no_auto|-no_automatic`
Shows all tasks that are directly or indirectly in the project's update properties (indirectly means the task is in a folder that is in the project's update properties).

`b|bl|baseline_project`
Shows the project's baseline.

`fol|folders`
Shows folders that are in the project's update properties.

`obj|objects`
Shows objects in the task that are either directly or indirectly in the project's update properties.

`t|tasks -auto|-automatic | -no_auto|-no_automatic`
Shows tasks that are directly in the project's update properties.

`tf|t_and_f|tasks_and_folders -auto|-automatic | -no_auto|-no_automatic`
Shows tasks and folders that are directly in the project's update properties.

Use the `-f` option to change the command's output format. Use `-u` to suppress the output's automatic numbering and `-ns` to suppress sorting. Use `-r` to show specified update properties recursively.

Use the `info` suboption to show information for the baseline project, tasks, and folders that make up the update properties for a project. Use `-r` with the `info`

suboption to show update properties recursively. Use `-v info` suboption to display detailed messages.

The default output format for `ccm up -show tasks` and `ccm up -show all_tasks` is:

```
Task %displayname: %task_synopsis
```

where:

`%displayname` is `%name` if DCM is not enabled, and <*database_ID*><*DCM_delimiter*>`%name` if DCM is enabled.
`%task_synopsis` is a description of the task.

When performing `ccm up -show tasks`, `ccm up -show all_tasks`, or `ccm up -show task_and_folder`, and if `show_auto_tasks` is set to `TRUE` in the `ccm.ini` file (Windows) or `.ccm.ini` file (UNIX), automatic tasks will display in the output.

The default output format for `up -show baseline_project` is:

```
%displayname
```

The default output format for `up -show folder` is:

```
Folder %displayname: %description
```

where:

`%displayname` is `%name` if DCM is not enabled, and <*database_ID*><*DCM_delimiter*>`%name` if DCM is enabled.
`%description` is the name of the folder.

The default output format for `up -show objects` is:

```
%objectname %status %owner %task
```

where:

`%objectname` is the object's *name-version:type:instance*.
`%status` is the status of the object.
`%owner` is the owner of the object.
`%task` is the task associated with the object.

The default output format for `up -show tasks_and_folders` is:

```
%displayname %description %task_synopsis
```

`-t|-task|-tasks` *`task_specs`*

Specifies the IDs of the tasks that you are adding or removing. For this argument's syntax, see Task specification.

`-u`

Suppresses automatic numbering of this command's output ("un-numbered").

`-vb|-valid_baselines`

Shows the possible project versions that could be used as a baseline for the project that is specified by *`project_spec`*.

Use the `-f` option to change the command's output format. Use `-u` to suppress automatic numbering of the output and `-ns` to suppress sorting. The default output format is `%displayname`.

`-v|-verbose`

Specifies that you want additional update properties messages output.

`-y`

Use `-y` to suppress confirmation messages. This option is useful for scripting.

Related topics

- update_properties examples

update_properties examples

View examples for the following operations:

- Add Tasks and/or Folders
- Compare Update Properties
- Remove Tasks and/or Folders
- Show Information for Baseline, Tasks, and Folders
- Set the Update Method
- Show/Set Valid Baselines
- Show Baseline, Tasks, Folders, and/or Objects

Add Tasks and/or Folders

- Add tasks 3,12-14, and 40-42 to the `utility-mary` project.

```
ccm update_prop -add -tasks 3,12-14,40-42 utility-mary

Adding tasks to utility-mary...
   Added task 3 to the update properties of project utility-mary
   Added task 12 to the update properties of project utility-mary
   Added task 13 to the update properties of project utility-mary
   Added task 14 to the update properties of project utility-mary
   Added task 40 to the update properties of project utility-mary
   Added task 41 to the update properties of project utility-mary
   Added task 42 to the update properties of project utility-mary

Added the following task(s) to project utility-mary
   Task 3
   Task 12
   Task 13
   Task 14
   Task 40
   Task 41
   Task 42
Added 7 tasks to utility-mary.
```

- In a DCM-initialized database, add tasks 5 and 17, imported from the `frankfurt` database, to the `utility-mary` project.

```
ccm update_prop -add -tasks frankfurt#5,frankfurt#17 utility-mary

Updating update properties of utility-mary...
   Added task 5
   Added task 17
Added 2 tasks.
```

- Add the folders 5-7 and 14 to the `utility-mary` project.

```
ccm update_prop -add -folders 5-7,114 utility-mary

Adding folders to utility-mary...
Added folder 5 to the update properties of project utility-mary
Added folder 6 to the update properties of project utility-mary
Added folder 7 to the update properties of project utility-mary
Added folder 114 to the update properties of project utility-mary

Added the following folder(s) to project utility-mary
      Folder 5
      Folder 6
      Folder 7
      Folder 114

Added 4 folders to utility-mary.
```

Compare Update Properties

- Compare the update properties for the `utility-mary` project and the `toolkit-2.0_gui` project.

```
ccm up -compare utility-mary utility-2.0

Baseline for utility-mary = utility-1.0
Baseline for utility-2.0 = utility-1.0
Tasks and Folders Only in Project utility-mary
     folder 1: All completed tasks for release 2.0
     folder 6: Assigned and completed tasks for mary for Release 2.0
     task   13: Insulated Development projects for mary for release 2.0
     task   14: Insulated Development products for mary for release 2.0

Tasks and Folders Only in Project utility-2.0
     folder 2: All tested tasks for release 2.0
     task   9: Integration Testing projects for release 2.0
     task   10: Integration Testing products for release 2.0

Tasks and Folders in Both Projects
```

Remove Tasks and/or Folders

- Remove tasks 3, 12-14, and 40-42 from the `new-mary` project.

```
ccm update_prop -remove -tasks 3,12-14,40-42 new-mary

Removing tasks from new-mary...
   Removed task 3 from the update properties of project new-mary
   Removed task 12 from the update properties of project new-mary
```

```
   Removed task 13 from the update properties of project new-mary
   Removed task 14 from the update properties of project new-mary
   Removed task 40 from the update properties of project new-mary
   Removed task 41 from the update properties of project new-mary
   Removed task 42 from the update properties of project new-mary

Removed the following task(s) from project new-mary
   Task 3
   Task 12
   Task 13
   Task 14
   Task 40
   Task 41
   Task 42
Removed 7 tasks from new-mary.
```

- Remove folders 5-7 and 114 from the `new-mary` project.

```
ccm up -rem -folders 5-7,114 new-mary

Removing folders from new-mary...
Removed folder 5 from the update properties of project new-mary
Removed folder 6 from the update properties of project new-mary
Removed folder 7 from the update properties of project new-mary
Removed folder 114 from the update properties of project new-mary
Removed the following folder(s) from project new-mary
      Folder 5
      Folder 6
      Folder 7
      Folder 114

Removed 4 folders from new-mary.
```

- In a DCM-initialized database, remove folder 37 imported from the `frankfurt` database, from the `utility-mary` project.

```
ccm up -rem -folders frankfurt#37 utility-mary

Updating update properties of utility-mary...
Removed folder 'frankfurt#37: Nepal's Tested Tasks'
Removed 1 folder.
```

Show Information for Baseline, Tasks, and Folders

- Show information recursively for the `utility-mary` project; view verbose messages.

```
ccm up -show info utility-mary -v -r
```

Set the Update Method

- Set the `utility-mary` project's update method to use a template instead of manual update properties.

```
ccm up -ru template utility-mary -r

utility-mary has been set to update using the process rule that will
maintain its update properties automatically.
```

Show/Set Valid Baselines

- Show the possible baselines of the `utility-mary` project.

```
ccm up -vb utility-mary

1) utility-3.0
2) utility-3.1
```

- Set the baseline of the `utility-mary` project to `utility-3.1`.

```
ccm up -mb utility-3.1 utility-mary

Set baseline to 'utility-3.1'
```

Show Baseline, Tasks, Folders, and/or Objects

- Show the baselines for the `utility-mary` project.

```
ccm up -sh bl utility-mary
```

- Show the tasks for the `utility-mary` project.

  The tasks shown are only those tasks included directly in the project's update properties.

```
ccm up -sh tasks utility-mary

1) Task 5: mary's insulated development projects
2) Task 6: mary's insulated development products
3) Task 40: Auto-calculation gives incorrect result
4) Task 53: Download of images occurs too slow
```

- Show the folders for the `utility-mary` project.

```
ccm up -sh folders utility-mary

1) Folder 111: mary's Insulated Development Task Folder
2) Folder 146: mary's Assigned Tasks
3) Folder 161: Tested Tasks for Release 3.2
```

- Show the tasks **and** folders for the `utility-mary` project.

The tasks shown are only those tasks included directly in the project's update properties.

```
ccm up -sh t_and_f utility-mary

1) 111 mary's Insulated Development Task Folder <void>
2) 146 mary's Assigned Tasks <void>
3) 161 Tested Tasks for Release 3.2 <void>
4) 40 <void> Auto-calculation gives incorrect result
5) 5 <void> mary's insulated development projects
6) 53 <void> Download of images occurs too slow
7) 6 <void> mary's insulated development products
```

- Show all tasks for the `utility-mary` project.

  The tasks shown are all of the tasks associated with the project, whether they are included directly in the project's update properties or are included indirectly through folders.

  ```
  ccm up -sh all_tasks utility-mary

  1) Task 15: Correct spelling errors in output
  2) Task 19: Rewrite messaging module
  3) Task 26: Close box no longer active
  4) Task 40: Auto-calculation gives incorrect result
  5) Task 5: mary's insulated development projects
  6) Task 53: Download of images occurs too slow
  7) Task 6: mary's insulated development products
  ```

- Show the objects for the `utility-mary` project.

  ```
  ccm up -show objects utility-mary

  1) main.c-4:csrc:1 integrate  mary  26
  2) main.h-3:incl:1 integrate  mary  26
  3) msg.c-5:csrc:1  integrate  mary  19
  4) msg.h-4:csrc:1  integrate   mary  19
  ```

update_template command

The `update_template` command is an alias for the process_rule command.

use command

Synopsis

```
ccm use [-t|-task task_number] [-r|-rules]
        file_spec [file_spec…]
ccm use [-r|-rules]
        -p|-project project_spec [project_spec…]
```

Description and uses

The `use` command performs either of the following operations:

- replaces an existing file, directory, or project with another version
- pastes an existing file, directory, or project that is not already in the current directory

> **Note** When you paste an object to a non-writable directory, a new directory version is checked out automatically.
>
> If you are in a shared project and your current directory is non-writable, the directory is checked out and associated automatically with the default (or specified) task and is checked in to the *integrate* state. You can disable the automatic check-in feature by setting `shared_project_directory_checkin` to `FALSE` in your initialization file. (See shared_project_directory_checkin.)

When you "use" a directory, the directory is updated automatically. You must check in the directory to make the changes in its content available to other users.

Options and arguments

file_spec

Specifies the object version(s) you are using.

-g

Brings up the appropriate dialog.

-p|-project *project_spec*

Adds a project to the current directory.

-r|-rules

Uses the version selected by the selection rules.

`-t|-task` *`task_number`*

Associates the newly checked-out directory with a task number if a `use` adds an object to a read-only directory.

Associates the object you add with the task. If the current (default) task is set and you do not specify a different task, the objects are associated with the current task automatically.

Examples

- Add an object that exists in the database, but not in your project. Added objects do not have to be writable by you.

  1. Query for the object version.

     ```
     ccm query -n objectname
     ```

  2. From the listed query items, identify the object version to use.

  3. Add the object version to your project.

     ```
     ccm use @item_number
     ```

- Add the `util-b2` and `tools-b2` projects to the current directory.

  ```
  ccm use -p util-b2 tools-b2
  ```

- Use version 2 of `display.c` in place of the current version.

  ```
  ccm use display.c-2
  ```

- Use the version of `clear.c` chosen by the selection rules.

  ```
  ccm use -rules clear.c
  ```

Related topics

- delete command
- unuse command

users command

Synopsis

```
ccm users
```

Description and uses

The `users` command brings up the `users` list for a specific Rational Synergy database. Using the default editor, you can add or remove users and their roles in this file.

When setting roles for users, be sure to set the first role to be the one the user will run in most often. (The first role is the default role when the user starts a session.) No user should have a first role setting of *ccm_admin*, even if he has access to it, for preventative reasons.

Database users are defined in a single table, one user per entry. One user list is used for the entire database. Add a user and his appropriate roles in the following format:

username = *role(s)*

A user name is followed by the roles to which the user has access. For example, a user could have one role or more than one role, as in the following example:

```
user bob      =  developer ccm_admin;
user john     =  writer;
user mary     =  developer build_mgr ccm_admin;
```

Note You must add a semicolon at the end of each **entry** in the `users` list, as shown in the example above.

Do not list a user twice, even if he has different roles in the same database. Instead, list all of his roles together, as shown in the example above (under `user erin` and `user matt`).

You must be in the *ccm_admin* role to use this command.

Options and arguments

None

Example

- Add a user to the database.
  1. Open the users list.

     ```
     ccm users
     ```

  2. Add a user to the database by copying an existing line, then replacing the new user's username.

3. Save and exit from the list.

- Change a user's roles in the database.
  1. Open the users list.

     ```
     ccm users
     ```

  2. Change a user's roles by adding or deleting roles after his name.

  3. Save and exit from the list.
- Delete a user from the database.
  1. Open the users list.

     ```
     ccm users
     ```

  2. Remove a user from a database by deleting the appropriate line.

  3. Save and exit from the list.

version command

Synopsis

```
ccm version [-d|-dbschema] [-a|-all] [-c|-ccm] [-i|-informix]
```

Description and uses

The `ccm version` command displays the version of Rational Synergy that is running.

Options and arguments

`-a|-all`

Displays the version of the current database schema, the INFORMIX database server, and the Rational Synergy release.

`-c|-ccm`

Displays the version of the Rational Synergy release.

If you enter `ccm version` without any switches, only the Rational Synergy release number is displayed.

`-d|-dbschema`

Displays the version of the database schema.

`-i|-informix`

Displays the version of the database server.

Example

Show the version of the current database schema, the INFORMIX database server, and the Rational Synergy release that are running.

```
ccm version -a
   INFORMIX Dynamic Server Version 10.00.UC5XA
   Rational Synergy Version 7.1
   Rational Synergy Schema Version 0111
```

view command

Synopsis

```
ccm view file_spec [file_spec...]
```

Description and uses

Use this command to view the source for an object that is not in the current directory or to invoke the viewer set by the cli.text_viewer option in your initialization file.

Options and arguments

file_spec

Specifies the object to be displayed.

Example

View version 8 of the `log.c` object.

```
ccm view log.c-8
```

Related topics

- cat command (UNIX only)
- edit command

work_area command

Synopsis

Show Work Area Options

```
ccm wa|work_area -show [-r|-recurse]
                 [-p|-project] project_spec [project_spec...]
```

Change Work Area Options

```
ccm wa|work_area [-wa|-maintain_wa] [-nwa|-no_wa]
                 [-cb|-copy_based] [-ncb|-not_copy_based]
                 [-rel|relative] [-nrel|not_relative]
                 [-mod|modifiable] [-nmod|not_modifiable]
                 [-wat|wa_time] [-nwat|no_wa_time]
                 [-tl|translate] [-ntl|no_translation]
                 [-r|-recurse] [-nr|-no_recurse]
                 [-setpath|-path|-set path]
                 [-pst|-project_subdir_template template]
                 [-p|-project] project_spec [project_spec...]
```

Find and Show All Projects with the Specified Character String in their Work Area Paths

```
ccm wa|work_area -find "find_string" [-reg|-regexp] [-replace "new_string"]
                -show
                [-scope working|prep|shared|checkpoint|STATIC|ALL|DB]
                [-p|-project] project_spec [project_spec...]
```

Find and Replace a Character String in a Work Area Path

```
ccm wa|work_area -find "find_string" [-reg|-regexp] -replace "new_string"
                 [-scope working|prep|shared|checkpoint|STATIC|ALL|DB]
                 [-new|-visible]
                 [-p|-project] project_spec [project_spec...]
```

Identify and Show All of the Projects with the Specified Database Path

```
ccm wa|work_area -dbpath "old_database_path"
                -show
                [-scope working|prep|shared|checkpoint|STATIC|ALL|DB]
                [-p|-project] project_spec [project_spec...]
                [-ns|-nosync]
```

Identify and Replace a Database Path

```
ccm wa|work_area -dbpath "old_database_path"
                 [-scope working|prep|shared|checkpoint|STATIC|ALL|DB]
                 [-p|-project] project_spec [project_spec...]
```

Description and uses

The `work_area` command enables you to show and modify work area options, and to show and update work area and database paths.

The following sections describe the ways in which you can change the work area information associated with one or more projects.

Changing work area options

The work area options you can change on a project are:

- whether the work area is maintained (synchronized to the file system): `-wa` or `-nwa`
- whether subprojects are relative to a parent project: `-rel` or `-nrel`
- whether the work area is copy-based or link-based: `-cb` or `-ncb`
- whether work area times are used: `-wat` or `-nwat`
- whether the source is copied as ASCII or binary files: `-tl` or `-ntl`
- the work area path (where the project is synchronized to the file system): `-setpath` *`path`* or `-pst` *`path`*
- whether files in the work area are not set to read-only, even if the object is static: `-mod` or `-nmod`

You can also set the project's work area path to a new location by changing the project-specific portion of the work area path. This change is made using the `-pst|-project_subdir_template` *`template`* option.

Updating an obsolete work area path

A project's work area path is the absolute path to the project in your file system. For example, a work area path could be:

Windows: `c:\ccm_wa\tools\ordtime-john`
UNIX: `/users/john/ccm_wa/tools/ordtime-john`

If you were to copy or unpack (`ccmdb cp` and `ccmdb unpack` on Windows) or (`ccmdb_cp` and `ccmdb_unpack` on UNIX) a database to a new database, it is likely that you would want to synchronize the new database's projects to new work area paths different from the work area paths to which the old database's projects were synchronized. This requires that you replace the old work area paths with new ones. The `-find "`*`find_string`*`" -replace "`*`new_string`*`"` options enable you to make these changes.

You can "rehearse" the change to the work area path by using the `-show` option.

Updating an obsolete database path

Each project has a database path associated with it. For example, a project's database path could be:

Windows: `y:\stanton\ccmdb\ccm_docs\db`
UNIX: `/vol/stanton/ccmdb/ccm_docs/db`

If you were to move a database to a new database path, the projects' database paths in the new location would be incorrect: they would still be set to the old path. Therefore, you must replace the old database paths with the new database path. The `-dbpath "old_database_path"` option enables you to make these changes.

You can "rehearse" the change to the database path by using the `-show` option.

Options and arguments

> **Note** All options default to the values set for the specified project.

`-cb|-copy_based`

Specifies that your work area will contain file copies rather than links. Work areas in Windows are copy-based.

`-dbpath "old_database_path"`

Windows:
Identifies the projects that have *`old_database_path`* in their `_ccmwaid.inf` (Rational Synergy work area identification) files, then changes those projects' `_ccmwaid.inf` file database strings to the path of the current database.

Use this option to update database paths when a database has been moved.

UNIX:
Identifies the projects that have *`old_database_path`* in their `.ccmwaid.inf` (Rational Synergy work area identification) files, then changes those projects' `.ccmwaid.inf` file database strings to the path of the current database.

Use this option to update database paths when a database has been moved.

You must have write access to the work area to use this option. You must be in the *ccm_admin* role to update the work area identification file of a non-writable project.

Note: For link-based work areas, the work area is resynchronized automatically in order to update the symbolic links to the new database. **Also, you must update all copy-based work areas before you update link-based work areas.** Copy-based work areas must be updated from a remote client (a session that was started with the `-rc` option).

After you update your work area path names, restart the session **without** the `-rc` option to update any link-based work areas.

`-find "`*`find_string`*`"`

Locates the projects that have *`find_string`* in their work area path names. Use this option to update work area paths that are out of date.

Use the `-reg` option if you want to use a regular expression (for example, `".*"`) in *`find_string`*. For more information, see Regular expressions.

You must have write access to the work area to use this option. You must be in the *ccm_admin* role to update the work area path name of a non-writable project.

On UNIX, **you must update all copy-based work areas before you update link-based work areas.** Copy-based work areas must be updated from a remote client (a session that was started with the `-rc` option). After you update your work area path names, restart the session **without** the `-rc` option to update any link-based work areas.

`-ncb|-not_copy_based`

Specifies that your work area contains links to files rather than copies. This option is not available for Windows.

`-new`

Used with `-find -replace` to indicate that this is a new database. This option is intended for two situations: when the work areas specified by the `-find` option are not visible, and when the work areas specified by the `-find` option are visible but should be ignored.

If this option is not specified, the command will operate only on projects with visible work areas. See the `-visible` option for information on updating visible work areas.

You can use this option only with the `-find` option.

`-nr|-no_recurse`

Do not recurse the project hierarchy when applying these options. Change only the specified project. This is the default.

`-nrel|-not_relative`

Do not make the work area relative to the parent project's work area, and on UNIX, use links for subprojects. This is the default when you first create a project.

`-ns|-nosync`

Specifies that your work area will not be synced.

`-ntl|-no_translation`

Specifies to copy source as binary files.

`-nwa|-no_wa`

Specifies that a work area not be maintained. Specifying this option will disconnect your work area from the database.

This option enables you to have a project that exists only in the database without a corresponding work area. (A project without a work area is called a "grouping project.") Use it to create logical groups of projects where a work area is not necessary or possible.

- For example, a work area is not necessary in the following example:

Suppose that you are the build manager and you need to create a project for a platform that does not exist now, but will exist in the near future. You know that you will need a platform for a new Windows OS soon, so you use this option to turn off the work area. Create the project. When you are ready to use the new platform project, you synchronize the new project to your work area by using the `-wa` option.

- A work area is not possible in the following scenario:

Suppose that you are developing software to be released on the Windows XP platform and the Solaris platform. The work areas for the projects for both of these platforms may not be visible from both platforms, but you want to view both projects, so you decide to group them. To create a project that groups both of these as subprojects, you would need to use a project without a work area.

`-nwat|-no_wa_time`

Specifies that new timestamps not be used, that is, sets the time stamps of the files to reflect the first modification time stored in Rational Synergy, rather than the time a file was copied to the work area.

`-p|-project` *`project_spec`*

Designates the project to which you want to apply the specified options.

You must specify *`project_name`*-*`version`* for the project to which you want to apply changes; however, you do not need to enter the option name `-p`.

`-pst|-project_subdir_template]` *`template`*

Changes the specified project's work area path (where the project is synchronized to the file system) to a new location. This parameter changes only the project-specific portion of the work area path. To change to a different part of the file system for your work area or synchronize your work area to a different platform, see -set|-path|-setpath path.

The default directory in which all project work areas are created is `ccm_wa` followed by the *`database_name`* in your home directory. By default, the project name and version are appended to the *`database_name`*. You can change the project-specific portion of the name to include `project_name`, `project_version`, `release`, `platform`, and `delimiter` by modifying the work area template.

If the previous path is visible to the interface host, it is moved to the new location. Otherwise, the work area is created when you execute the `work_area` command with this option.

`-r|-recurse`

Causes all projects in the project hierarchy to be updated along with the specified project.

`-reg|-regexp`

Indicates that *`new_string`* and *`find_string`* are regular expressions.

You can use this option only with the `-find` option.

`-rel|-relative`

Makes the work area path relative to the parent project's path. A relative work area can be used in multiple projects, as long as the project is non-modifiable. It can be used in only one project if it is modifiable.

`-replace "`*`new_string`*`"`

Substitutes *`new_string`* for *`find_string`* in the work area paths of all of the projects found using the `-find` *`find_string`* option.

Use the `-reg` option if you want to use a regular expression (for example, ".*") for *`new_string`*.

You can use this option only with the `-find` option.

`-scope`

Establishes an initial criterion for which projects can be found using the `-find` or `-dbpath` options.

The scope option can be one of the following states:

working (all of your *working*-state projects)
checkpoint (all of your *checkpoint*-state projects)
prep (all *prep* projects)
shared (all *shared* projects)

or one of the following case-sensitive keywords:

`STATIC` (all of your non-writable projects; for example, projects in the *integrate*, *test*, *sqa*, or *released* state)
`ALL` (all projects, regardless of status)
`DB` (all projects for the current database, regardless of ownership or status)

If the scope is *working*, *checkpoint*, or `ALL`, the projects must be owned by you.

The default scope is *working*.

You can use this option only with the `-find` or `-dbpath` option.

`-set|-path|-setpath` *path*

Changes the specified project's work area path to the new location. This parameter changes the non-project-specific portion of the work area path. To change the project-specific portion of the name, such as `project_name`, `project_version`, `release`, `platform`, and `delimiter` by modifying the work area template, see -pst|-project_subdir_template] template.

The default directory in which all project work areas are created is the database name in `ccm_wa` in your home directory. This is your *work area path*. If you want to use a different part of the file system for your work area or you want to synchronize your work area to a different platform, you can change the work area path using this option.

If the previous path is visible to the interface host, it is moved to the new location. Otherwise, the work area is created when you execute the `work_area` command with this option.

You can change the work area path of a read-only project only if you are in the *ccm_admin* role.

`-show`

Shows the work area options for the specified project(s) if you are not using the `-find` or `-dbpath` option. If you are using the `-find` or `-dbpath` option, you are shown the projects that a) satisfy the scope criterion, are specified by the `-p` option, or are in the selection set, and b) contain the specified string or database path.

When you use this option with `-replace`, you are shown what would result if you were to perform the operation.

`-visible`

Indicates that only visible work areas should be considered for update.

This is the default. A message is displayed for work areas skipped because the work area is not visible to the interface.

See the `-new` option for information on updating work area paths of projects that are in a database in which work areas have not yet been created.

You can use this option only with the `-find` option.

`-wa|-maintain_wa`

Maintain a work area. Setting this option synchronizes the work area and keeps it synchronized.

To stop a sync from the CLI, enter `<CTRL+C>` at any time.

However, if you stop the sync, you will receive an error message stating that errors may occur in your work area. The errors will not occur until you try to use the work area. To avoid problems, perform a complete synchronization of the work area before you use it.

You can use this option on a read-only project only if you are in the *ccm_admin* role.

`-wat|-wa_time`

Specifies that new timestamps be used, that is, sets the timestamps of the files to reflect the time a file was copied to the work area, rather than the Rational Synergy modification time.

This setting should be used if you are using a third-party Make tool. If you do not use this option, the work area maintains the modification time of a file. While at first this may sound appropriate, problems arise when a different version of a file is selected into the work area. If the file's modification time is older than any product in this work area that depends on it, the Make tool will not be able to determine that the product needs to be rebuilt.

Related topics

- work_area examples
- delimiter command
- reconcile command
- resync command
- sync command

work_area examples

View examples for the following operations:

- Show Work Area Options
- Change Work Area Options
- Find and Replace a Character String in a Work Area Path
- Identify and Replace a Database Path

Show Work Area Options

- Show the work area options for the `ico_may2-1` project.

  ```
  ccm work_area -show -p ico_may2-1
  ```

Change Work Area Options

- Change the path of the specified project and resynchronize the project.

  Windows: `ccm work_area /setpath c:\users\linda\new_wa\database /p ico_feb1-1`
  UNIX: `ccm work_area -setpath /users/linda/new_wa/database -p ico_feb1-1`

Find and Replace a Character String in a Work Area Path

- Find all of your *working* projects with visible work areas whose paths contain the "`-`" character, and change the "`-`" to a "`~`".

  ```
  ccm wa -find "-" -replace "~"
  ```

  This command would be appropriate for database delimiter changes. It would need to be executed by each user from enough sessions to cover all work areas. If all of a user's work areas could be seen from one session, one session would be sufficient. However, if a user has both Windows and UNIX work areas, this command would need to be run from both Windows and UNIX clients.

- Find all of the *prep* projects with visible work areas whose paths contain the "`-`" character and change the "`-`" to a "`~`".

  ```
  ccm wa -find "-" -replace "~" -scope prep
  ```

  This would be appropriate for database delimiter changes. It would need to be executed by a build manager from enough sessions to cover all build management work areas. If all of the *prep* work areas could be seen from one session, one session would be sufficient. However, if there are both Windows and UNIX build management work areas, this command would need to be run from both Windows and UNIX clients.

- In a newly copied database that still has work area paths from the original database, find all of your work area paths that contain the string `platform` and change the string to `services`.

  ```
  ccm wa -find platform -replace services -new
  ```

  This would be appropriate for a database that was unpacked or copied to a new name. The -`new` option is used here to create all new work areas. This is because the old work areas are for the old database. If the old work areas are to be reused, such as for a database that is moved, the user would first need to update the work area ID files using the -`dbpath` option so that the old work areas would appear to be for this database.

- On Windows, using a regular expression, shorten the work area path `c:\users\joe\+joe45\hsaw~1` to `c:\users\joe45\hsaw~1` by removing the `\+joe45` directory. (Note the leading special character.)

  ```
  ccm wa -find "\+joe45\\\\" -reg -replace "" -p hsaw~1
  ```

Identify and Replace a Database Path

Find all of the user's working projects with work areas for the database "`/vol/acrel5/ccmdb/ccm_platform`," and update the work area ID files with the path to the current database.

```
ccm wa -dbpath "/vol/acrel5/ccmdb/ccm_platform"
```

This would be appropriate for a database that has been moved to a new location in the file system, and the old work areas are still available. It would need to be executed by each user from enough sessions to cover all work areas. If all of a user's work areas could be seen from one session, one session would be sufficient. If, however, a user has both Windows and UNIX work areas, this command would need to be run from both Windows and UNIX clients.

Default

You can set the wa_path_template and project_subdir_template options in your `ccm.ini file (Windows), or .ccm.ini` file (UNIX), or use the work_area command.

wa_snapshot command

The `wa_snapshot` command is an alias for the copy_to_file_system command.

Learn more about

The following pages contain advanced, in-depth information about using Rational Synergy.

- Conflict detection

  Conflict Detection identifies the types of conflicts detected by Rational Synergy.
- Date formats

  Date Formats describes how to set and change formats for the date and time in Rational Synergy.
- Defining the merge tool

  Define the Merge Tool explains how to use your own merge tool instead of the default merge tool.
- Migration rules

  Migration Rules explains the migration rules, and gives detailed examples.
- Query expressions

  Query Expressions provides details on how to construct query expressions.
- Relationships

  Relationships describes the object-to-object relationships used in Rational Synergy.
- Shared projects

  Shared Projects introduces the concepts and uses of shared projects.
- SOAD scopes

  SOAD Scopes explains how the Save Offline and Delete (SOAD) tool uses scopes to create a list of objects to save offline and delete.
- Triggers

  Triggers explains how to define programs that are used to notify users about activity in the database, such as when objects change state or problems are submitted.
- Work area

  Work Area explains absolute versus relative work areas and copy-based versus link-based work areas, and gives examples.

- Work area conflicts

  Work Area Conflicts describes the different types of work area conflicts you may encounter and explains how work area conflicts can be resolved.

Conflict detection

Conflict detection is Rational Synergy's criteria and process for warning you that your work area or project has somehow deviated from the expected. Rational Synergy's conflict detection keeps track of your project's tasks and history relationships to ensure that your project contains all of the correct members; however, if any deviations appear necessary, Rational Synergy warns you so that you can decide how to proceed. Many different types of conflicts are possible, such as:

- If you are working as a developer, you may have conflicts when your work area is out of sync with the database, or when a file, directory, or project has parallel versions. For specific information about resolving conflicts while working in Rational Synergy, see the online help.
- If you are working as a build manager, you may have conflicts between your project's members and its update properties.

The best time to show conflicts is immediately after an update, because the project members are up-to-date with the project's update properties at that time.

If your update properties contain any folders that use queries, you can update the folder contents after your last update. Since discrepancies between your project's update properties and your project's members display as conflicts, your project might show additional conflicts if you have not updated recently.

The following topics are discussed:

- Conflicts detected by Rational Synergy
- Task and object relationships
- Types of conflicts
- Large-scale conflict detection

Conflicts detected by Rational Synergy

You can view conflicts while working from the command line or one of the graphical user interfaces. When you have a conflict, you will see the object versions and associated tasks that have conflicts, and a brief description of each object's conflict. The conflict messages are defined in the following table.

The table below, and the explanation of conflict detection that follows the table, use these definitions:

- "conflict" is defined as one of the following situations:

  A file associated with a task not specified to be in your project was included.

  A file associated with a task that was specified to be in your project was not included.

  A file's task relationships are not as expected (i.e., no task or multiple tasks were associated with the object).
- "explicit" means "directly requested," that is, included in your project's update properties.
- "implicit" means "indirectly depended upon or partially included," that is, not included in your project's update properties.

Conflict message	Conflicts shown by default?	Description
No task	yes	The object version is included implicitly in this project, but not associated with any task. (It cannot be included explicitly because this would require its task to be included in the project's update properties.)
Multiple Tasks	no	The object version is included in this project, and is associated with multiple tasks.
Implicitly included	yes	The object version is not explicitly specified, but it is included in the project.
Included by "use" operation?	yes	The object version is not explicitly specified, not implicitly required, and update would not have selected it.
Implicitly required but before baseline	no	The object version is implicitly required, but it is a predecessor of the baseline. (This is not really a conflict, since it is implicitly included, but it might indicate a process problem.)

Conflict message	Conflicts shown by default?	Description
Implicitly required but not included - newer	yes	The object version is implicitly required, but is not included in the project. It is a successor of the current selected version of that object.
Implicitly required by multiple tasks - newer	yes	The object version is implicitly required because it was associated with a task that was implicitly included because another file in the project was associated with multiple tasks. The object version in conflict is not included in the project, and is a successor of the version of that file currently in the project.
Implicitly required but not included - parallel	yes	The object version is implicitly required but not included in the project. It is parallel to the currently selected version, and may require a merge.
Implicitly required by multiple tasks - parallel	yes	The object version is implicitly required because it is associated with a task that was implicitly included because another file in the project was associated with multiple tasks. The object version in conflict is not included in the project, and is parallel to the version of the file currently in the project.
Explicitly specified but before baseline	no	The object version is explicitly specified on the project, but is a predecessor of the baseline. (This is not really a conflict, since it is implicitly included, but it might indicate a process problem.)
Explicitly specified but not included - newer	yes	The object version is explicitly specified on the project, but is a successor of the currently selected version of that object.
Explicitly specified but not included - parallel	yes	The object version is explicitly specified on the project, but is not included in the project. It is parallel to the current version, and may require a merge.

Conflict message	Conflicts shown by default?	Description
Explicitly specified but object not in project	no	The object version is explicitly specified on the project, but no versions of it are in the project. This is probably normal, since the same update properties are shared across entire project hierarchies.
Implicitly required but object not in project	no	The object version is implicitly required through a task included in the project, but no versions of it are in the project. This is probably normal, since the same update properties are shared across entire project hierarchies.
Conflicts detected for tasks if shown with task option (displayed as messages)		
Implicitly included	yes	The task is implicitly included in the project.
Implicit task from explicit object	yes	The task's associated file has multiple assigned tasks. At least one of the object's associated tasks is explicit (that is, included in the update properties), but this task is not.
Implicitly required but not included	yes	The task is implicitly required, but not included in the project.
Explicitly specified but not included	yes	The task is explicitly specified by the project, but is not included.
Explicitly specified but no files in project	no	The task is explicitly specified by the project, but none of its files are included in the project. This is probably normal, since the same update properties are shared across entire project hierarchies.
Excluded task explicitly included	yes	An excluded task is included in a baseline and tasks in the project's project grouping.
Excluded task implicitly included	yes	An excluded task is implicitly included in a project's update properties.
Completed fix task not included	yes	A bad task is included in a project's update properties, but its `completed` good task is not included.

Conflict message	Conflicts shown by default?	Description
Assigned fix task not included	no	A bad task is included in a project's update properties, but its `task_assigned` good task is not included.
Task fixed by task not included	yes	A bad task is not included in a project's update properties, but the good task is included.

Task and object relationships

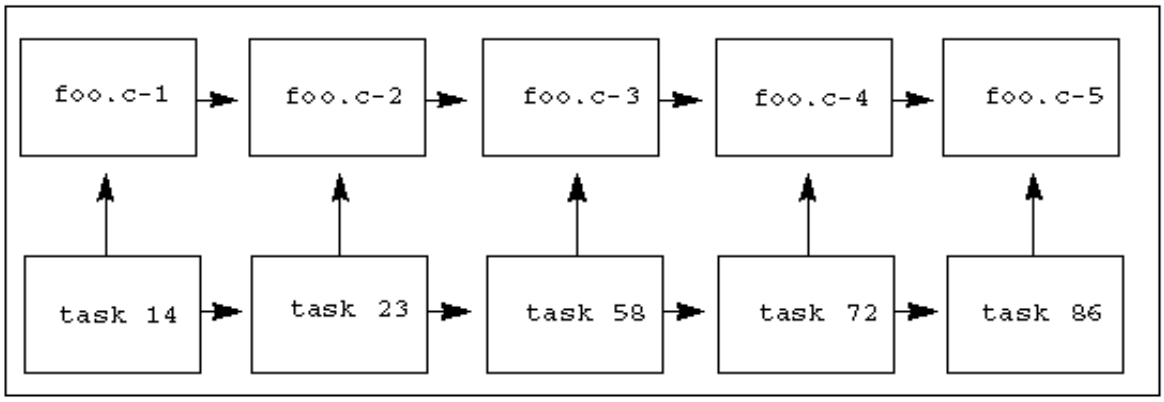
A task and a set of object versions can have a relationship. In Rational Synergy, you can associate a set of object versions with a task, which tells Rational Synergy that those object versions should be used together and that they depend on each others' changes. If your project includes only part of the changes associated with a task, your project probably will fail to build, or even worse, will fail to run correctly.

For example, if you change a function signature, you must update every other program that calls the function to have the signature change. You must include all of those changes together in the project, or you should not include any of the changes.

Object history relationships

Tasks have a history relationship, but it is different from an object's history relationship. An object's history usually is numerically consecutive. A task's history is a conceptual relationship only, based on its associated files' history relationship. Because a task groups files necessary to complete a change, the task's history relationship causes a current set of changes to depend on the previous set of changes.

The following figure shows a version history for one file and each task associated with the file throughout the object's history.



The `foo.c` file has five versions. Each version is associated with a different task. (The task number associated with each version is shown below the object version.)

Rational Synergy considers that a change to an object version contains the changes to all of its predecessor object versions. Therefore, in the example shown above, version 3 is considered to contain the changes from versions 2 and 1.

For example, if you changed the signature of a function in version 2, then version 3, version 4, and every version after that would include that signature change. The changes are layered on top of each other. Even a change that removes a part of another change is layered on top of its history versions. However, the change in `foo.c-3` applies only to `foo.c-2`, since the change was made to that version.

Task dependencies

Furthermore, because version 3 contains the changes from versions 1 and 2, version 3's associated tasks are considered to depend on the tasks associated with versions 1 and 2. So in this example, task 58 depends on tasks 23 and 14.

Explicitly specified update properties

Consider the project `myproj-bill`, which contains `foo.c-4`.

Recall that a project's update properties contain a baseline project and a list of tasks. (It can also contain task folders, but they are collections of tasks.) If a task is specified in the project's update properties (either directly or by using a task folder), the project has **specified explicitly** that it should include the files associated with that task. For example, if the `myproj-bill` project's update properties include tasks 72 and 23, then it has specified explicitly that it should include the object versions associated with tasks 72 and 23. In the previous figure, if the project has specified explicitly tasks 72 and 23, then it also has specified explicitly object versions `foo.c-4` and `foo.c-2`.

Remember that `foo.c-4` contains the changes from `foo.c-2`, and task 72 depends on task 23.

When you update a project, its explicitly specified object versions are its candidates. Update will select the most appropriate candidate, usually the newest. So in this example, `myproj-bill` would use tasks 72 and 23 to determine the candidate list: `foo.c-4` and `foo.c-2`. It would pick `foo.c-4` as the newest candidate. Therefore, the project would include both the changes from `foo.c-4` and `foo.c-2`. Likewise, it would contain the changes from both task 72 and task 23.

Implicitly specified update properties

Because the `myproj-bill` project contains `foo.c-4`, it contains task 72, which its update properties explicitly specified. It also depends on `foo.c-3`, since `foo.c-3` is a predecessor of `foo.c-4`. It also depends on the task that is associated with `foo.c-3`: task 58.

However, if task 58 (and therefore, `foo.c-3`) is not explicitly specified in `myproj-bill`'s update properties, but the change is included anyway through its history relationship, then both the task and the object version are **specified implicitly** in the project. Note that files associated with an implicitly specified task are not included in the project automatically.

Types of conflicts

Suppose that you are preparing to release your software application. You specify that the release should contain tasks 72 and 23, but you do not specify task 58. After your build, you might be surprised to find that task 58 was included in the application you were preparing. Rational Synergy can warn you that a task you did not request will be included. This is called a *conflict*.

There are many different types of conflicts. If you had manually used `foo.c-5` in the project, but the project's update properties didn't explicitly specify task 86, and no other tasks that you had explicitly specified depended on task 86, that would be another type of conflict. Rational Synergy can warn you that it appears that an object version has been used in your project, rather than its task being explicitly specified.

Some conflicts are more serious than others.

For example, your team might decide that they will not fix more than one bug in a single version of a file because then it is too difficult to tell which changes fixed which bug. Further, your team decides that each developer should associate only one task with each object version he changes. If an object version in the release you are preparing is associated with more than one task, you need to know so that you can remind the developer that this is not a good idea. However, it is not a severe conflict, since the software you are preparing for the release contains all the changes it needs.

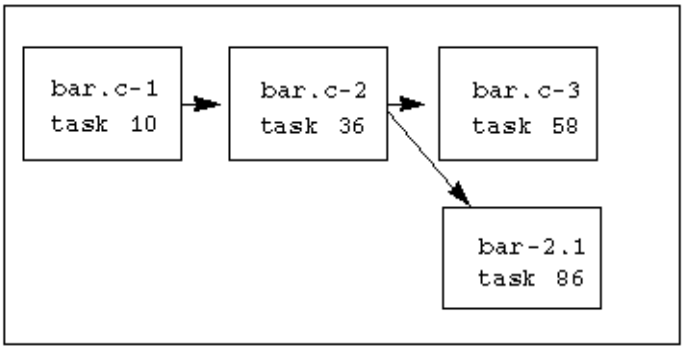
A more serious conflict is an implicitly included object version that is not associated with **any** task.

Rational Synergy can warn you about both types of conflicts.

Parallel conflicts

One of the most important types of conflict detection is detecting parallel object versions.

If your project has explicitly specified a change and it is not included, that is a serious conflict. For example, consider a situation where two parallel files are associated with two different tasks, and both tasks are explicitly specified. In this example, assume that `myproj-bill` contains `bar.c-3`. The `bar.c` file has the history relationships and task associations shown in the following figure.



The `myproj-bill` project's update properties specify that tasks 58 and 86 should be included. But your project includes only `bar.c-3`, which is associated with task 58. It is impossible for it also to include `bar.c-2.1`, which is associated with task 86, because that is a parallel branch. There is no one version of `bar.c` that contains both changes you have requested. This is a severe conflict, since a version that you know should be included in your project is not there.

Parallel conflicts can mean missing changes, but there are other types of missing changes as well.

Missing changes

Consider what would happen if you manually included (with `ccm use` or from the Use dialog) `bar.c-2.1` in the `myproj-bill` project. Changes for both tasks 58 and 86 would be missing, even though they were explicitly specified, because they are newer than the version of the file currently in the project.

This change is explicitly specified, but missing. You might notice that it is missing if you looked at the tasks included in your project's update properties to see that they were included in your project. Other types of conflicts are even more difficult to detect.

Let's say we updated the update properties on `myproj-bill` to include tasks 86 and 36, rather than tasks 86 and 58. Now task 58 is no longer explicitly specified. Task 86 is associated with `foo.c-5`, and its predecessor is `foo.c-4`, which is associated with task 72. So task 86 implicitly includes task 72. If your project includes `foo.c-5`, it includes both changes, and everything is fine. But what about `bar.c`? `bar.c-2.1` is specified explicitly because it is associated with task 86, and `bar.c-3` is specified implicitly because it is associated with task 58 (which is implicitly included in your project). Therefore, once again there is no version of `bar.c` that contains all of the changes that your project requested.

Large-scale conflict detection

Now let's consider a project with hundreds of members, each of which has many versions in its history, and a release that comprises hundreds of tasks. No matter how careful your team is, the bigger your project, the greater the opportunity for errors, for example, those due to parallel development (forgotten merges) or human error (forgotten object/task association). The solution is to find out about the errors and correct them **before** you build. Rational Synergy can detect conflicts on a large-scale project so that your team can resolve issues before they become problems (i.e., hold up the build).

Rational Synergy uses its knowledge of all of the history relationships and task relationships to detect these conflicts for you. In all, it can detect 24 different types of conflicts, although only 16 are shown by default. The other 8 are not severe, and they are not shown by default because seeing extra conflicts makes it more difficult to concentrate on the conflicts that really affect your software's integrity. However, if you want to change the type of conflicts that are shown by default, see conflict_parameters. Note that you must be in the *ccm_admin* role to change these default settings.

Rational Synergy analyzes a project to determine whether it has any conflicts, then displays the conflicts. The operation is available for a single project rather than a hierarchy, although you can show conflicts for a project hierarchy from the command line (`ccm conflicts`).

Depending on the size and characteristics of your project, conflict detection can be time-consuming, so only you will know the best time to show conflicts. Build managers should show conflicts after every update of a *prep* project. Developers may not want to show conflicts unless they suspect that their projects contain parallel versions or other conflicts that are causing problems.

Date formats

The following topics explain how to set and manipulate the date and time formats displayed in Rational Synergy dialogs and commands, reports, and date/time input formats to various functions.

- Dates displayed by Rational Synergy
- Dates accepted as Input by Rational Synergy
- Setting environment variables
- General information

Dates displayed by Rational Synergy

Rational Synergy displays dates in several Rational Synergy dialogs and `ccm` commands. By default, these dates are displayed in a locale-specific format, such as "Fri Oct 30 12:04:38 1998", or "Mittwoch, 6. Mai 1998, 17:13:20 Uhr".

The following are the rules for date conversion on output:

1. If you set the environment variable `CCM_DATETIME_FMT`, Rational Synergy uses `strftime()` with the format specified in that variable. Note that most, but not all, date/time conversion is done on the engine; to get the effect of a format everywhere, you must set the environment variable for both interface and engine processes.
2. If you do not set the environment variable, and the process is running on UNIX, then `strftime()` is used with a format of `%c`. Note that this includes UNIX engines for Windows clients. This format produces the locale-specific date and time format referred to in the first paragraph above.
3. If you do not set the environment variable, and the process is running on Windows, the long date format in the Regional Settings control panel is used, together with the time format defined by Windows for the locale. Note that there is a control panel setting for the date format, but not for the time format.

Dates accepted as Input by Rational Synergy

Dates are accepted as input when using the time ("xxxxx") operator in queries, when setting the values of time attributes using the `ccm attr` command, and by certain other commands such as `ccm clean_cache`.

On some UNIX clients, dates can be accepted during input in locale-specific formats. Locale-specific date and time input is not available on some other UNIX clients, nor on Windows; on these systems, dates and times may be displayed in a locale-specific format, but must be entered in a numerical format with U.S. ordering. The systems that support locale-specific input vary from release to release, and are listed in the Rational Synergy README. You can disable the locale-specific date input on any system by setting the environment variable `CCM_NO_LOCALE_TIMES`.

On those systems that support it, locale-specific date input is handled by the routine `strptime()` with a format of `%c`. Rational Synergy also tries a few other formats - `%c`, `%Ec`, `%c %Z`, `%x %X`, `%X %x`, `%x %X %Z`, `%X %x %Z`, `%x`, and if not on Solaris, `%X`.

If `CCM_NO_LOCALE_TIMES` is set, or if the system does not support `strptime()`, Rational Synergy performs its own date and time translation. Rational Synergy tries several formats, most of which are specific to the U.S. conventions in language, punctuation, and ordering.

A format that should be acceptable for input on any system, locale-specific or not, is `YYYY/MM/DD hh:mm:ss`. Alternatively, use the relative date and time format. Use the relative date/time format in queries and with commands such as `ccm clean_cache`.

Relative dates and times are specified in the following format:

```
-d:h:m:s
+d:h:m:s
```

Relative dates and times are interpreted relative to the start of today. Some sample date and time formats follow:

```
'Thu May 2 12:15'
'April 10 1998'          (The time defaults to the start of the day)
'2001/08/21 11:36:15'    (August 21st 2001 at 11:36:15am)
'-0:0:0:0'               (start of today)
'+0:0:0:0'               (midnight tonight)
'-1:18:0:0'              (6 A.M on the day before yesterday)
'-2:0:0:0'               (start of day before yesterday)
'-2:8:0:0'               (4pm three days ago)
```

Setting environment variables

You can set environment variables for the client (interface process) by setting the variables before calling `ccm start`.

To set an environment variable for the engine on Windows, add a section `[Engine environment variables]` to your `ccm.ini` file, and add the required variables and their values to that section.

To set an environment variable for the engine on UNIX, edit the script `$CCM_HOME/bin/util/ccm_engine`.

Additionally, you can set a Rational Synergy command, `ccm env`, to set the interface or engine environment variables dynamically. The `ccm env` command is not installed by default, but it is available in the `extras\contrib` (Windows) or `extras/contrib` (UNIX) directory in your Rational Synergy installation area.

General information

Acceptable dates are from 1 second after the start of January 1st 1970 (GMT) until the end of 2037. For example, in the Pacific Standard Time time zone, the earliest acceptable time is 1 second after 4pm on December 31st 1969.

> **Note Do not** assume that a date displayed by Rational Synergy can be converted back into exactly the same time. The underlying OS code (even when using `CCM_NO_LOCALE_TIMES`) is not always accurate to the second, and conversions between time zones or between daylight saving times and standard time are known to be erratic on many systems.

On some releases of some operating systems, `strptime()` is available but does not work correctly. In these cases (for example, HP-UX 11.0), set the `CCM_NO_LOCALE_TIMES` environment variable to any value.

Date formats using ISO 8601 format

Date/time strings used in queries can be written in the ISO 8601 format of "2006-08-21 T09:12:15-0100" using the following parameters:

- The full four-digit year number must be supplied.
- The month, day, hour, minute, and seconds field must be present and must be exactly two digits, with a leading zero if necessary.
- The year, month, and day fields must be separated by a single dash (-).
- The date and time must be separated by a single upper-case T
- The time fields must be separated by a single colon (:)
- The time zone field must be present, and can be either a single upper-case Z for UTC (GMT) time, or a plus (+) or minus (-) character followed by an offset from UTC
- The offset from UTC is in hours (two digits) and minutes (two digits); the hours and minutes may be separated by an optional colon (:) separator. Thus, -0500 and -05:00 are both valid time zone fields.

Defining the merge tool

You can redefine or change only the interactive merge and compare tool. The following sections tell you how to do so.

- Prerequisites
- Command keywords
- Where to define the merge and compare commands

Prerequisites

The new merge tool must meet the following requirements:

- It must be installed on your system.
- The merge command must be able to accept three arguments: "ancestor" (the last common file before the multiple versions), and *file1* and *file2* (the files to be merged).
- The command must have a "save" capability that allows you to save to another file.

If your interactive merge tool does not have an output file option, you must save the output to the location specified by Rational Synergy. A message appears in the Merge dialog and Message View during the merge, telling you where Rational Synergy will save the file. By default, the file is saved as `%file1` (the name of the first file specified) in your system's default temporary directory. For example, if `%file1` is `foo.c-32` and the temporary directory is `c:\temp (Windows) or /tmp (UNIX)`, Rational Synergy will save your merge results to `c:\temp\foo.c (Windows) or /tmp/foo.c (UNIX)`. Following the save, Rational Synergy copies the contents of the saved file to the new, controlled file.

If the save directory already contains a file with the name you specified, Rational Synergy generates the file name `merge_#.out`, where "`#`" is an integer incremented by 1 for each file saved.

Command keywords

Use the keywords shown in the following table when defining your interactive merge or compare tool. These keywords are expanded during the merge or compare; enter them exactly as shown.

Keyword	Description
`%outfile`	output file where merge output is saved
`%ancestor`	common ancestor file, if one exists
`%file1`	first file selected to merge
`%file2`	second file selected to merge
`%file1_label`	name and version of the first file
`%file2_label`	name and version of the second file

Where to define the merge and compare commands

You can use a different interactive merge tool by making the following change:

- In the type definition for your object

  You can change a specific object type's `merge_cmd` attribute by using the typedef command.

  Changes made to a type affect only objects of that type.

Migration rules

Migration rules use file characteristics to map files in the file system to Rational Synergy database objects when you perform a migrate operation. Rational Synergy also uses the migration rules when you perform create, reconcile, and build operations. Use the appropriate information:

- Migrating - Windows operating systems
- Migrating - UNIX operating systems

Migrating - Windows operating systems

The following sections explain more about the migration rules:

- Migration rules files
- Auto-generated rules
- Rule syntax and meaning
- Rules file example
- Migrating binary archive files
- Troubleshooting the migration rules

Migration rules files

You can define migration rules in any ASCII file. The default rules are in the *database_path*\lib\Windows\migrate.rul file.

Use an alternative migration rules file, with an arbitrary name, in any of the following ways:

- Create a rules file, then enter its name in a system or personal ccm.ini file using the following syntax:

```
migrate.options.rules_file: alternative_rules_filename
```

- Create a rules file, then use the Migrate Options dialog in Synergy Classic to load the rules file.
- Create a rules file, then use the following syntax on the command line to load a rules file:

```
ccm migrate -rules alternative_rules_filename
```

If you define more than one alternative migration rules file, the files' precedence is as follows (highest to lowest):

1. File specified in the Migrate Options dialog in Synergy Classic or ccm migrate command option
2. File specified in the personal ccm.ini file
3. File specified in the system ccm.ini file
4. Default rules

Auto-generated rules

By default, the default migrate rules file contains an INCLUDE_AUTO_RULES directive. This directive includes automatically generated rules defined on type definitions for the type of client that is currently running. For example, you may have defined an msword type with two patterns that match .doc and .dot suffixes on a Windows client. The auto-generated rules will include the following two rules at the point where the INCLUDE_AUTO_RULES directive has been specified:

```
MAP_FILE_TO_TYPE .*[Dd][Oo][Cc]$ msword
MAP_FILE_TO_TYPE .*[Dd][Oo][Tt]$ msword
```

The benefit of using auto-generated rules is that the type-based rules are defined on the corresponding type definition. Moreover, if the type definition is exported from one database and imported to another, its corresponding migrate rules accompany it. This is especially useful if you are using DCM and replicating type definitions between databases.

The migrate rules file completely controls the ordering of migrate rules. If desired, you can remove the `INCLUDE_AUTO_RULES` directive so that the migrate rules file explicitly defines every rule. You can also move the position of the directive in the file to change the point at which the auto-generated rules are included.

The Auto-Generated rules are produced using the following process:

1. Auto-generated rules from the predefined `ascii` and `binary` types are produced first.
2. Each type that uses `ascii` or `binary` as its super type is then processed, so that the inheritance tree of types is traversed from top to bottom and left to right. This means that more specific types that are children of some super type generate rules after their parent super type.
3. Each non-comment entry in the type's file match list generates a rule of the form:

   `MAP_FILE_TO_TYPE` *`regular_expression type`*
4. Each type that has Ignore on Migrate set to `TRUE` generates a rule of the form:

   `MAP_TYPE_TO_IGNORE` *`type`* `TRUE`

You can view the auto-generated rules that would be included by an `INCLUDE_AUTO_RULES` directive as follows:

From the CLI, use the `ccm show -mar` command.

Rule syntax and meaning

Each rule in a migration rules file sets an attribute on, or performs an operation on, files with the specified characteristics.

The rule syntax is as follows:

`mapping operand_1 operand_2`

- *`mapping`* is the mapping action performed.

  The mapping action sets object attributes or performs ignore or collapse operations.
- *`operand1`* is the file characteristic.

  The file characteristic is a regular expression, a string, or a macro, depending on the mapping action.
- *`operand2`* is the value used by the mapping action.

  The value is `TRUE` or `FALSE` for ignore and collapse mapping actions, or the attribute value for the other mapping actions.

The rules are applied in the order in which they appear in the rules file. A pound sign precedes comments.

> **Note** You can use upper, lower, or mixed case in file names, depending on which Windows operating system you are using.

For example, the following rules set the object type to `csrc` and version to "`1.0`" for files ending in "`.c`":

```
# C source files
MAP_FILE_TO_TYPE .*\.c$ csrc
MAP_TYPE_TO_VERSION csrc 1.0
```

> **Note** Back slash (\ ) separators in path names are converted to forward slash (/) separators when the migration rules are applied. Therefore, migration rules can contain standard regular expression escapes (\ ).

The following table shows the possible migration rules and their operands.

Mapping	Operand 1	Operand 2
to TYPE		
`MAP_FILE_TO_TYPE`	string or regexp	valid type
`MAP_TYPE_TO_TYPE`	string or regexp	valid type
`MAP_MODE_TO_TYPE`	value or macro	valid type
to VERSION		
`MAP_FILE_TO_VERSION`	string or regexp	valid version
`MAP_TYPE_TO_VERSION`	string or regexp	valid version
`MAP_MODE_TO_VERSION`	value or macro	valid version
to IGNORE		
`MAP_FILE_TO_IGNORE`	string or regexp	`TRUE` or `FALSE`
`MAP_TYPE_TO_IGNORE`	string or regexp	`TRUE` or `FALSE`
`MAP_MODE_TO_IGNORE`	value or macro	`TRUE` or `FALSE`
to COLLAPSE		

Mapping (Continued)	Operand 1	Operand 2
MAP_FILE_TO_COLLAPSE	string or regexp	TRUE or FALSE
MAP_TYPE_TO_COLLAPSE	string or regexp	TRUE or FALSE
MAP_MODE_TO_COLLAPSE	value or macro	TRUE or FALSE
to ATTRIBUTE		
MAP_FILE_TO_ATTRIBUTE	string or regexp	TRUE or FALSE
MAP_KEY_TO_ATTR	string	string

The following sections show how the rules are used:

- Map to TYPE
- Map to VERSION
- Map to IGNORE
- Map to COLLAPSE
- Map to ATTRIBUTE

Map to TYPE

The MAP_*_TO_TYPE rule sets type attributes. Usually this is the first rule you enter for a file because the type attribute can be used in subsequent rules.

If the object type is %expand_pvcs, migrate treats the matching file as a PVCS archive and the migrate operation extracts the deltas from the file. These deltas become objects that appear in the list of objects being previewed.

> **Note** The MAP_MODE_TO_* rules accept one pre-defined macro for *operand_1*: %dir.

Examples

- Set the Rational objects' type attributes to csrc for all files with a .c extension.

```
MAP_FILE_TO_TYPE .*\.c$ csrc
```

- Expand all PVCS archive files by uncommenting the following rule.

```
MAP_FILE_TO_TYPE   .*\...[Vv]$ %expand_pvcs
```

- Set the Rational objects' type attributes to dir for all directories.

```
MAP_MODE_TO_TYPE %dir dir
```

Map to VERSION

The MAP_*_TO_VERSION rule sets the version attributes.

> **Note** The MAP_MODE_TO_* rules accept one pre-defined macro for *operand_1*: %dir.

Example

- Set the Rational objects' version attributes to 2 for all files with the .c extension.

```
MAP_FILE_TO_VERSION .*\.c$ 2
```

> **Note** If an instance of this object already exists with a version set to 2, the next appropriate version is used.

Map to IGNORE

The MAP_*_TO_IGNORE rule, when set to TRUE, causes files to be ignored.

> **Note** If a directory is marked as ignored, all files in the directory hierarchy are ignored.

Examples

- Ignore files with a .map file extension.

```
MAP_FILE_TO_IGNORE .*\.map$ TRUE
```

- Ignore files of type makefile.

```
MAP_TYPE_TO_IGNORE makefile TRUE
```

- Recognize PVCS files (by un-commenting commented-out, default migration rules).

  These rules are usually commented out so that files with suffixes ending in V or v (such as .wav files) are not ignored. The second rule causes unexpanded PVCS files to be ignored.

```
MAP_FILE_TO_TYPE   .*\...[Vv]$ %expand_pvcs
MAP_FILE_TO_IGNORE .*\...[Vv]$ TRUE
```

Map to COLLAPSE

The MAP_*_TO_COLLAPSE rule, when set to TRUE, causes directories to be collapsed before they are migrated.

All the matching directory's children become its parent directory's children. This method is typically used to bring archive files out of their archive directories and up to the level of their checked-out versions.

> **Caution** For migrates, all COLLAPSE and %expand rules are evaluated during Preview only; they are not re-evaluated if the rules are edited.

Example

- Collapse `tmp` directories.

```
MAP_FILE_TO_COLLAPSE .*/tmp$ TRUE
```

Map to ATTRIBUTE

The `MAP_*_TO_ATTRIBUTE` rule sets the specified attribute to the specified value.

> **Note** You can map only text attributes to Rational Synergy attributes.

Example

- Set the SYNERGY objects' `reviewer` attributes to the files' PVCS Extended Revision Attributes, `reviewed_by`.

```
MAP_KEY_TO_ATTR reviewed_by reviewer
```

> **Note** If the archive attribute does not exist on the objects, Rational Synergy creates the attribute as type `text`.

Rules file example

Suppose you are using the following migration rules file (the numbers at the left are provided for reference):

```
1. MAP_FILE_TO_TYPE .*\.mk$ makefile
2. MAP_FILE_TO_TYPE [Mm]akefile[^/]*$ makefile
3. MAP_TYPE_TO_IGNORE makefile TRUE
4. MAP_FILE_TO_TYPE .*\...[Vv]$ %expand_pvcs
5. MAP_FILE_TO_IGNORE .*\...[Vv]$ TRUE
6. MAP_MODE_TO_TYPE %dir dir
```

Then, suppose you are migrating the following files into the Rational Synergy database:

```
dir1                <dir>
Makefile.joe
Makefile.jov
```

The first file, `dir1`, is a directory. The rule applied to this file is:

```
6. MAP_MODE_TO_TYPE %dir dir
```

The second file, `Makefile.joe`, is a makefile. The rules applied to this file are:

```
2. MAP_FILE_TO_TYPE [Mm]akefile[^/]*$ makefile
3. MAP_TYPE_TO_IGNORE makefile TRUE
```

Rule 2 is applied because the pattern "`[Mm]akefile[^/]*$`" matches the makefile's file name, `Makefile.joe`. Rule 3 is applied because the file object has been given the type

makefile. The result is that files of type makefile are ignored. You also could write the following new rule to achieve the same result:

```
MAP_FILE_TO_IGNORE [Mm]akefile[^/]*$
```

The third file, Makefile.jov, is a PVCS archive. The rules applied to this file are:

```
4. MAP_FILE_TO_TYPE .*\...[Vv]$ %expand_pvcs
5. MAP_FILE_TO_IGNORE .*\...[Vv]$ TRUE
```

Rule 4 is applied because the "*\...[Vv]$" pattern matches the file name, Makefile.jov. This rule causes the migrate operation to extract all the deltas inside the archive file and create an object for each of them in the preview list. Rule 5 has the same pattern match and marks the archive file as **ignored**. The result is that new object versions are created for the expanded PVCS files, but the archive file itself is ignored. Also, rules are applied to each file expanded from an archive file; therefore, each Makefile.jov file is assigned the type makefile according to rule **3.**

> **Note** Be sure to test your migration rules on test files before applying them to production files.
>
> For example, the MAP_FILE_TO_* rules match a pattern to a file name. Because a file name includes the entire path, the MAP_FILE_TO_TYPE makefile rule would map a directory such as C:\my_makefiles\foo.c to type makefile, because the pattern "makefile" is in the path. Therefore, the rule MAP_FILE_TO_TYPE /makefile$ makefile (using a slash before the file name, and anchored to the end of the path) would be a better rule to use to ensure that the rule is applied to files, not to directories.

Migrating binary archive files

When assigning types to archive files, the migrate operation first extracts files from the archive files, then applies the migration rules to the resulting file name. If the resulting file name does not match any existing type, it could be assigned to ascii automatically. If the archive contents are not actually ascii, the migrate operation cannot load the file.

To prevent a migrate operation from loading archive files as ascii, do one of the following:

- Add rules mapping binary archive file names to a non-ascii type, such as binary (which is the usual way).
- Manually map the binary archive files to a non-ascii type for items selected from the Preview Results list.
- Check the Preview results before performing the load to ensure that the types are correct.

Troubleshooting the migration rules

If the rules that you have entered have no effect, do the following:

1. Check that the changed rules have been saved and that the edit rules session has ended.
2. Check that the rules have not been superseded by rules that follow them in the rules list.
3. Consider that `MAP_*_TO_TYPE`, which uses `%expand_pvcs`, is applied **during a preview, not after it is edited** when other rules are re-applied.
4. Consider that `MAP_FILE_TO_*` rules match a full file path. For example, if a directory by the name of `bitmap` exists and a rule is `MAP_FILE_TO_TYPE bitmap binary`, the directory and everything in its hierarchy would be set to type `binary`. To avoid this behavior, set the first operand to `/bitmap$`.

   For example, to migrate `bitmap` files as type `binary`, and `bitmap` directories as type `dir`, use the following rules:

   ```
   MAP_FILE_TO_TYPE /bitmap$ binary
   MAP_MODE_TO_TYPE $dir dir
   ```

Migrating - UNIX operating systems

The following sections explain more about the migration rules:

- Migration rules files
- Auto-generated rules
- Rule syntax and meaning
- Rules file example
- Migrating binary archive files
- Troubleshooting the migration rules

Migration rules files

You can define migration rules in any ASCII file. The default rules are in the *database_path*/lib/UNIX/migrate.rul file.

Use an alternative migration rules file, with an arbitrary name, in any of the following ways:

- Create a rules file, then enter its name in a system or personal ccm.ini file using the following syntax:

  migrate.options.rules_file: *alternative_rules_filename*

- Create a rules file, then use the Migrate Options dialog to load the rules file.
- Create a rules file, then use the following syntax on the command line to load a rules file:

  ccm migrate -rules *alternative_rules_filename*

If you define more than one alternative migration rules file, the files' precedence is as follows (highest to lowest):

1. File specified in the Migrate Options dialog or ccm migrate command option
2. File specified in the personal .ccm.ini file
3. File specified in the system .ccm.ini file
4. Default rules

Auto-generated rules

By default, the default migrate rules file contains an INCLUDE_AUTO_RULES directive. This directive includes automatically generated rules defined on type definitions for the type of client that is currently running. For example, you may have defined an msword type with two patterns that match .doc and .dot suffixes on a Windows client. The auto-generated rules will include the following two rules at the point where the INCLUDE_AUTO_RULES directive has been specified:

```
MAP_FILE_TO_TYPE .*[Dd][Oo][Cc]$ msword
MAP_FILE_TO_TYPE .*[Dd][Oo][Tt]$ msword
```

The benefit of using auto-generated rules is that the type-based rules are defined on the corresponding type definition. Moreover, if the type definition is exported from one database and imported to another, its corresponding migrate rules accompany it. This is especially useful if you are using DCM and replicating type definitions between databases.

The migrate rules file completely controls the ordering of migrate rules. If desired, you can remove the `INCLUDE_AUTO_RULES` directive so that the migrate rules file explicitly defines every rule. You can also move the position of the directive in the file to change the point at which the auto-generated rules are included.

The Auto-Generated rules are produced using the following process:

1. Auto-generated rules from the predefined `ascii` and `binary` types are produced first.
2. Each type that uses `ascii` or `binary` as its super type is then processed, so that the inheritance tree of types is traversed from top to bottom and left to right. This means that more specific types that are children of some super type generate rules after their parent super type.
3. Each non-comment entry in the type's file match list generates a rule of the form:

   `MAP_FILE_TO_TYPE` *`regular_expression type`*
4. Each type that has Ignore on Migrate set to `TRUE` generates a rule of the form:

   `MAP_TYPE_TO_IGNORE` *`type`* `TRUE`

You can view the auto-generated rules that would be included by an `INCLUDE_AUTO_RULES` directive as follows:

From the CLI, use the `ccm show -mar` command.

Rule syntax and meaning

Each rule in a migration rules file sets an attribute on, or performs an operation on, files with the specified characteristics.

The rule syntax is as follows:

`mapping operand_1 operand_2`

- *`mapping`* is the mapping action performed.

  The mapping action sets object attributes or performs ignore or collapse operations.
- *`operand1`* is the file characteristic.

  The file characteristic is a regular expression, a string, or a macro, depending on the mapping action.
- *`operand2`* is the value used by the mapping action.

  The value is `TRUE` or `FALSE` for ignore and collapse mapping actions, or the attribute value for the other mapping actions.

The rules are applied in the order in which they appear in the rules file. A pound sign precedes comments.

For example, the following rules set the object type to `csrc` and version to "`1.0`" for files ending in "`.c`":

```
# C source files
MAP_FILE_TO_TYPE .*\.c$ csrc
MAP_TYPE_TO_VERSION csrc 1.0
```

The following table shows the possible migration rules and their operands.

Mapping	Operand 1	Operand 2
to TYPE		
MAP_FILE_TO_TYPE	string or regexp	valid type
MAP_TYPE_TO_TYPE	string or regexp	valid type
MAP_MODE_TO_TYPE	value or macro	valid type
to VERSION		
MAP_FILE_TO_VERSION	string or regexp	valid version
MAP_TYPE_TO_VERSION	string or regexp	valid version
MAP_MODE_TO_VERSION	value or macro	valid version
to IGNORE		
MAP_FILE_TO_IGNORE	string or regexp	TRUE or FALSE
MAP_TYPE_TO_IGNORE	string or regexp	TRUE or FALSE
MAP_MODE_TO_IGNORE	value or macro	TRUE or FALSE
to COLLAPSE		
MAP_FILE_TO_COLLAPSE	string or regexp	TRUE or FALSE
MAP_TYPE_TO_COLLAPSE	string or regexp	TRUE or FALSE
MAP_MODE_TO_COLLAPSE	value or macro	TRUE or FALSE
to STATUS		
MAP_STATE_TO_STATUS	string	valid value
to ATTRIBUTE		

Mapping (Continued)	Operand 1	Operand 2
MAP_FILE_TO_ATTRIBUTE	string or regexp	TRUE or FALSE
MAP_KEY_TO_ATTR	symbols (lower-case character string)	string

The following sections show how the rules are used:

- Map to TYPE
- Map to VERSION
- Map to IGNORE
- Map to COLLAPSE
- Map to STATUS
- Map to ATTRIBUTE

Map to TYPE

The MAP_*_TO_TYPE rule sets type attributes. Usually this is the first rule you enter for a file because the type attribute can be used in subsequent rules.

If the object type is %expand_rcs or %expand_sccs migrate treats the matching file as an RCS or SCCS archive and the migrate operation extracts the deltas from the file. These deltas become objects that appear in the list of objects being previewed.

> **Note:** The MAP_MODE_TO_* rules accept a positive integer, an octal value, or a predefined macro as their first parameter. The positive integer or octal value represents a file mode. An octal value must begin with a zero; for example, a file with rwxr--r-- permissions is matched with an octal value of 0100744. The 0100000 of the bit pattern is the code for a regular file.
>
> The predefined macros %dir, %link, and %exec are less restrictive than the integer or octal values. Whereas a rule such as MAP_MODE_TO_TYPE 0100111 executable matches only a file with the permissions ----x--x--x, the rule MAP_MODE_TO_TYPE %exec executable matches a file with any of the executable permissions set (for example, -rwx--x--x)

Examples

- Set the Rational objects' type attributes to csrc for all files with a .c extension.

```
MAP_FILE_TO_TYPE .*\.c$ csrc
```

- Set the objects' `type` attributes to `executable` for all files with mode `executable`. (`%exec` evaluates to `TRUE` if any of the file's permissions—user, group or other—are executable).

  ```
  MAP_MODE_TO_TYPE %exec executable
  ```

- Set the Rational objects' `type` attributes to `dir` for all directories.

  ```
  MAP_MODE_TO_TYPE %dir dir
  ```

Map to VERSION

The `MAP_*_TO_VERSION` rule sets the `version` attributes.

> **Note:** The `MAP_MODE_TO_*` rules accept a positive integer, an octal value, or a predefined macro as their first parameter. The positive integer or octal value represents a file mode. An octal value must begin with a zero; for example, a file with `rwxr--r--` permissions is matched with an octal value of 0100744. The 0100000 of the bit pattern is the code for a regular file.
>
> The predefined macros `%dir`, `%link`, and `%exec` are less restrictive than the integer or octal values. Whereas a rule such as `MAP_MODE_TO_TYPE 0100111 executable` matches only a file with the permissions `----x--x--x`, the rule `MAP_MODE_TO_TYPE %exec executable` matches a file with any of the executable permissions set (for example, `-rwx--x--x`).

Examples

- Set the Rational objects' `version` attributes to 2 for all files with the `.c` extension.

  ```
  MAP_FILE_TO_VERSION .*\.c$ 2
  ```

  > **Note** If an instance of this object already exists with a version set to 2, the next appropriate version is used.

- Set the `version` attributes to `ver1` for all files of type `link`.

  ```
  MAP_MODE_TO_VERSION %link link
  ```

- Set the `version` attributes to `pre1` for all files that are user read- or write-only.

  ```
  MAP_MODE_TO_VERSION 600 pre1
  ```

Map to IGNORE

The `MAP_*_TO_IGNORE` rule, when set to `TRUE`, causes files to be ignored.

Note If a directory is marked as ignored, all files in the directory hierarchy are ignored.

Examples

- Ignore all files in the `tmp` directory.

  ```
  MAP_FILE_TO_IGNORE /tmp/ TRUE
  ```

- Ignore files with an SCCS file extension.

  ```
  MAP_FILE_TO_IGNORE .*/s\.[^/]+$ TRUE
  ```

- Ignore files of type `makefile`.

  ```
  MAP_TYPE_TO_IGNORE makefile TRUE
  ```

Map to COLLAPSE

The `MAP_*_TO_COLLAPSE` rule, when set to `TRUE`, causes directories to be collapsed before they are migrated.

All the matching directory's children become its parent directory's children. This method is typically used to bring archive files out of their archive directories and up to the level of their checked-out versions.

Caution For migrates, all `COLLAPSE` and `%expand` rules are evaluated during Preview only; they are not re-evaluated if the rules are edited.

Example

- Collapse `SCCS` directories.

  ```
  MAP_FILE_TO_COLLAPSE .*/SCCS$ TRUE
  ```

Map to STATUS

The `MAP_STATE_TO_STATUS` rule sets the `status` attributes.

Note: You must have the *ccm_admin* role to use this rule, and either set Use Status From Archive ON in the GUI or use the `-meta_status` option on the `migrate` command line.

This mapping is valid only when migrating files from an RCS archive.

Example

- Set the objects' `status` attributes to *`released`* for all files with RCS status `Exp`.

  ```
  MAP_STATE_TO_STATUS Exp released
  ```

Map to ATTRIBUTE

The `MAP_*_TO_ATTRIBUTE` rule sets the specified attribute to the specified value.

> **Note** You can map only text attributes to Rational Synergy attributes.

Example

- Set the objects' `release` attributes to the files' RCS `symbols` attribute.

```
MAP_KEY_TO_ATTR symbols release
```

> **Note** If the archive attribute does not exist on the objects, Rational Synergy creates the attribute as type `text`.

Rules file example

Suppose you are using the following migration rules file (the numbers at the left are provided for reference):

```
1. MAP_MODE_TO_TYPE %exec executable
2. MAP_FILE_TO_TYPE .*\.mk$ makefile
3. MAP_FILE_TO_TYPE [Mm]akefile[^/]*$ makefile
4. MAP_TYPE_TO_IGNORE makefile TRUE
5. MAP_FILE_TO_TYPE .*/s\.[^/]+$ %expand_sccs
6. MAP_FILE_TO_IGNORE .*/s\.[^/]+$ TRUE
7. MAP_FILE_TO_COLLAPSE .*/SCCS$ TRUE
8. MAP_MODE_TO_TYPE %dir dir
```

Then, suppose you are migrating the following files into the Rational Synergy database:

```
dir1              <dir>
Makefile.joe      -rw-r--r--
SCCS/             drwxr-xr-x
s.Makefile.joe    -r--r--r--
```

The first file, `dir1`, is a directory. The rules applied to this file are:

```
1. MAP_MODE_TO_TYPE %exec executable
8. MAP_MODE_TO_TYPE %dir dir
```

Rule **1** is applied because `a/` has an executable file bit set ("`x`"). Rule **8** is applied because the directory file bit is set. The resulting file type is `dir` because the second rule overrides the first rule.

The second file, `Makefile.joe`, is a makefile. The rules applied to this file are:

```
3. MAP_FILE_TO_TYPE [Mm]akefile[^/]*$ makefile
4. MAP_TYPE_TO_IGNORE makefile TRUE
```

Rule **3** is applied because the pattern "`[Mm]akefile[^/]*$`" matches the makefile's file name, `Makefile.joe`. Rule **4** is applied because the file object has been given the type `makefile`. The result is that files of type `makefile` are ignored. You also could write the following new rule to achieve the same result:

```
MAP_FILE_TO_IGNORE [Mm]akefile[^/]*$
```

The third file, `SCCS/`, is an `SCCS` directory. The rule applied to this file is:

```
7. MAP_FILE_TO_COLLAPSE .*/SCCS$ TRUE
```

(Rule **8** is applied also, but the file is already collapsed by rule **7.**)

The first rule is applied because the pattern "`.*/SCCS$`" matches the directory's name. The rule causes the `SCCS` directory not to be migrated and the directory's member files to be migrated into the migration's destination directory.

The fourth file, `s.Makefile.joe`, is an SCCS archive. The rules applied to this file are:

```
5. MAP_FILE_TO_TYPE .*/s\.[^/]+$ %expand_sccs
6. MAP_FILE_TO_IGNORE .*/s\.[^/]+$ TRUE
```

Rule **5** is applied because the "`.*/s\.[^/]+$`" pattern matches the file name, `s.Makefile.joe`. This rule causes the migrate operation to extract all the deltas inside the archive file and create an object for each of them in the preview list. Rule **6** has the same pattern match and marks the archive file as **ignored**. The result is that new object versions are created for the expanded SCCS files, but the SCCS files themselves are ignored. Also, rules are applied to each file expanded from an archive file; therefore, each `Makefile.joe` file is assigned the type `makefile` according to Rule **4.**

> **Note** Be sure to test your migration rules on test files before applying them to production files.
>
> For example, the `MAP_FILE_TO_*` rules match a pattern to a file name. Because a file name includes the entire path, the `MAP_FILE_TO_TYPE` makefile rule would map a directory such as `/user/mark/proj1/my_makefiles/foo.c` to type `makefile`, because the pattern "`makefile`" is in the path. Therefore, the rule `MAP_FILE_TO_TYPE /makefile$ makefile` (using a slash before the file name, and anchored to the end of the path) would be a better rule to use to ensure that the rule is applied to files, not to directories.

Migrating binary archive files

When assigning types to archive files, the migrate operation first extracts files from the archive files, then applies the migration rules to the resulting file name. If the resulting file name does not match any existing type, it could be assigned to `ascii` automatically. If the archive contents are not actually `ascii`, the migrate operation cannot load the file.

To prevent a migrate operation from loading archive files as `ascii`, do one of the following:

- Add rules mapping binary archive file names to a non-`ascii` type, such as binary (which is the usual way).
- Manually map the binary archive files to a non-`ascii` type for items selected from the Preview Results list.
- Check the Preview results before performing the load to ensure that the types are correct.

Troubleshooting the migration rules

If the rules that you have entered have no effect, do the following:

1. Check that the changed rules have been saved and that the edit rules session has ended.
2. Check that the rules have not been superseded by rules that follow them in the rules list.
3. Consider that `MAP_*_TO_TYPE`, which uses `%expand_sccs` and `%expand_rcs`, is applied **during a preview, not after it is edited** when other rules are re-applied.
4. Consider that `MAP_FILE_TO_*` rules match a full file path. For example, if a directory by the name of `core` exists and a rule is `MAP_FILE_TO_TYPE core binary`, the directory and everything in its hierarchy would be set to type `binary`. To avoid this behavior, set the first operand to `/core$`.

   For example, to migrate `core` files as type `binary` and `core` directories as type `dir`, use the following rules:

   ```
   MAP_FILE_TO_TYPE /core$ binary
   MAP_MODE_TO_TYPE $dir dir
   ```

Query expressions

The following topics explain how to construct Rational Synergy database queries.

- Types of queries
- Query clause elements
- Sample queries

Types of queries

Queries, or *query expressions*, are created using the `ccm query` command with one or more *query clauses*. Query clauses are the individual search criteria that make up a query expression.

The following sections show how to construct query expressions using different types of query clauses.

- Queries using attribute value clauses
- Queries using function test clauses
- Queries using both attribute value and function test clauses
- Queries using keywords
- Nested queries

Queries using attribute value clauses

A query clause based on an attribute value finds all object versions with (or without) matching attributes.

The syntax for this type of clause consists of the attribute name (*attr_name*), a relative operator (*relative_operator*), and the attribute's value (*constant*), as shown below:

"*attr_name relative_operator 'constant'*"

Examples

- Find all object versions with a status of *working*.

  ```
  ccm query "status='working'"
  ```

  The built-in shortcut for this command is as follows:

  ```
  ccm query -s working
  ```

- Find and show all object versions with version 2.

  ```
  ccm query "version='2'"
  ```

  The built-in shortcut for this command is as follows:

  ```
  ccm query -v 2
  ```

Queries using function test clauses

A query clause based on a function test finds all object versions that match the function's results.

The syntax for this type of clause consists of the function (*function*) and its arguments, as shown below:

"*function('function_arguments')*"

The functions are pre-defined. See Function definitions for the functions' descriptions.

Examples

- Find and delete all object versions that have the predecessor `ico-1:executable:2`.

  ```
  ccm query "has_predecessor('ico-1:executable:2')"
  ccm delete @
  ```

- Find and select all object versions of type `wdt`.

  ```
  ccm query "type='wdt'"
  ```

  The built-in shortcut for this command is as follows:

  ```
  ccm query -type wdt
  ```

Queries using both attribute value and function test clauses

You can combine query clauses to narrow your search. The following lines show how to combine query clauses:

"**not** *query_clause*"
"*query_clause* **and** *query_clause*"
"*query_clause* **or** *query_clause*"

Examples

- Find and show all object versions that are not members of a project.

  ```
  ccm query "not is_bound()"
  ```

- Find and show all object versions that are members of a project, and that have a modification time older than December 12, 2001.

  ```
  ccm query "is_bound() and modify_time < time('Fri Dec 12 2001')"
  ```

Queries using keywords

Certain *keywords* relative to time can be used in query expressions. The following table shows the valid keywords.

Keyword	Description
`%today_begin`	Beginning of today, 00:00:00
`%today_end`	End of today, 23:59:59
`%this_week_begin`	End of this week, 23:59:59) See Note 1
`%this_week_end`	Beginning of last week, 00:00:00 See Note 1
`%last_week_begin`	Beginning of last week, 00:00:00 See Note 1

Keyword	Description
%last_week_end	Beginning of last week, 23:59:59 See Note 1
%this_month_begin	Beginning of this month, 00:00:00
%this_month_end	End of this month, 23:59:59
%last_month_begin	Beginning of last month, 00:00:00
%last_month_end	End of last month, 23:59:59
%this_year_begin	1 Jan this year, 00:00:00
%this_year_end	31 Dec this year, 23:59:59
%today_minus<N>days	Today minus <N> days, 00:00:00
%today_plus<N>days	Today plus <N> days, 00:00:00
%today_minus<N>weeks	Today minus <N> weeks, 00:00:00
%today_plus<N>weeks	Today plus <N> weeks, 00:00:00
%today_minus<N>months	Today minus <N> months, 00:00:00 See Note 2
%today_plus<N>months	Today plus <N> months, 00:00:00 See Note 2
%today_minus<N>years	Today minus <N> years, 00:00:00 See Note 3
%today_plus<N>years	Today plus <N> years, 00:00:00 See Note 3

Note 1 The first day of the week is Sunday by default. The model attribute `start_day_of_week` can be set to change this default. A value of 1 means Monday, 2 Tuesday and so on.

Note 2 When subtracting or adding months, if the current day of the month is greater than the number of days in the resultant month, the effective date is the last day of that month. In both cases, time is 00:00:00. For example, if today was 30 January 2003, then `%today_plus1month` would be 28 February 2003, 00:00:00.

Note 3 When subtracting or adding years, if the current day of the month is greater than the number of days in that month in the resultant year, the effective date is the last day of the month. For example, if today was 29 February 2004, then `%today_plus1years` gives 28 February 2005, 00:00:00.

Example

- Show all files called `file1.c` that were created today.

```
ccm query "name='file1.c' and create_time > time('%today_begin')"
```

Nested queries

A *nested query* is a query expression that uses a function test, in which one or more of the function's arguments is a query expression.

Query functions usually have the following syntax:

```
query_function('object_name'|
               'project_name'|
                  'type_name'|
                  'attr_name'|
             'privilege_name'
                 [,sort_order])
```

You can replace any object name, project name, or type name argument with any query expression that evaluates to zero or more objects of the appropriate type. Queries can be nested to any desired depth.

Examples

- Find all members of all projects named `editor`.

```
ccm query "is_member_of(cvtype='project' and name='editor')"
```

- Find all version 1.0 projects that have the same members as project `toolkit-1.0`.

```
ccm query "has_member(is_member_of('1/project/toolkit/1.0')) and
version = '1.0'"
```

- Find all subprojects in all projects named `editor`, using the fastest search method (by specifying "none").

```
ccm query "hierarchy_asm_members(cvtype='project' and name = 'editor',
'none')"
```

- Find all objects that have object `save.c-1` (of type `csrc`) as their predecessor.

```
ccm query "has_predecessor(cvtype='csrc' and name='save.c' and
version='1')"
```

- Find all objects used in directory `sources-1` in project `editor-fcheng`.

```
ccm query "is_child_of('sources-1:dir:1', cvtype='project' and
name='editor' and version='fcheng')"
```

- Find all projects containing objects associated with tasks, for which the tasks' release values are set to 1.0.

```
ccm query "has_member(is_associated_cv_of(cvtype='task' and
release='1.0'))"
```

Note When you construct a query expression in the Query dialog's Query field, you need not enclose the outer query expression in double quotes.

Query clause elements

Query clauses are made of individual elements. You can use the following elements to construct a query clause:

- Functions
- Relative operators
- Logical operators
- Constants
- Grouping Query Clauses

Functions

Use the following function arguments and functions to construct function-based query clauses.

- Function arguments
- Function definitions

Function arguments

The function arguments are as follows:

attr_name

Specifies the name of any attribute, such as `is_product` or `platform`.

object_name

Specifies the object reference form of any object version:

`name-version``:`*`type`*`:`*`instance`*

order_spec

Specifies the search order. If the value `depth` is used for the *`order_spec`*, the value indicates that a depth-first search is done. `breadth` is used to specify a breadth-first search. If the value "none" is used for the *`order_spec`*, it indicates that the order is not significant and the search can be done in any order (the fastest method is used).

```
none | depth | breadth
```

privilege_name

Specifies the name of a permission such as *read* or *write*.

project_name

Specifies the name of any project object version:

`project_name-version`

Function definitions

The query functions are as follows:

baseline (’*baseline_spec***’)**

Queries the database for baselines that match the specified information.

build (’*build_string***’)**

Queries the database for baselines that have the specified build string. This query function expands to "`cvtype='baseline`' and `build='build_string'"`, which returns the set of all baselines with the given build string, i.e., with a matching build attribute.

cr (’*cr_id***’)**

Queries the database for change requests that have the specified number.

folder (’*folder_id***’)**

Queries the database for folders that have the specified number.

has_attr (’*attr_name***’)**

Queries the database for all object versions that have the attribute *`attr_name`* (for example, `is_product` or `platform`).

has_child (’*object_name***’,’***project_name***’)**

Queries the database for all directory object versions in *`project_name`* that have *`object_name`* as a member.

has_member (’*object_name***’)**

Queries the database for all project object versions that have the specified object version as a member.

has_model (’*object_name***’)**

Queries the database for all of the object versions that use the specified model object version as their model.

For example, if you use the Base Model, this query is `has_model`(`base-1:model:base’`).

has_no_*relationship* **()**

Queries the database for objects that do not have a relationship of that name to any object.

For example, `has_no_successor` returns every object that does not have a successor.

has_predecessor (’*object_name***’)**

Queries the database for all object versions that have the specified object version as an immediate predecessor.

has_priv ('*privilege_name*')

privilege_name specifies the name of a privilege, such as *read* or *write*.

has_purpose ('*purpose_name*')

Queries the database for all projects that have the specified purpose.

has_*relationship* ('*objectspec*', '*operator*', *time*)

Queries the database for all objects having the specified relation to the specified object whose relation create time matches the specified operator (that is, =, !=, >,<=, >, or >=) and time value. For example, Rational Synergy uses has_successor to show history relationships.

The *relationship* can be any established relationship such as associated_cv or associated_task.

Note that has_*relationship*('*objectspec*', '*operator*', *time*) is the inverse query of has_*relationship*_of('*objectspec*', '*operator*', *time*).

For information on how to create a relationship, see the history command.

has_type ('*type_name*')

Queries the database for all object versions of type *type_name*. For example, a query of has_type ('csrc-1:cvtype:base') might find HelloWorld-1:csrc:1.

hierarchy_project_members ('*project_name*', *order_spec*)

Queries the database for all projects in the project hierarchy specified by *project_name*. The *order_spec* argument specifies the search order, as described under recursive_is_member_of.

The query returns an ordered list of object version names. Use of other queries in conjunction with this query may change the result's order.

Note that the *project_name* **is** returned from this query.

For a description of *order_spec*, see order_spec.

is_bound()

Queries the database for object versions that are members of any project. This is best used when specifying other limiting options, such as the name of a project.

is_child_of('*object_name*','*project_name*')

Queries the database for all object versions that are members of directory *object_name* in project *project_name*.

is_hist_leaf()

Queries the database for objects that are leaf nodes when viewing history (that is, queries for objects that do not have successors).

is_hist_root ()

Queries the database for objects that are root nodes when viewing history (that is, queries for objects that do not have predecessors).

is_member_of (’*project_name***’)**

Queries the database for all object versions that are members of the specified project.

is_model_of (’*object_name***’)**

Queries the database for the model object version associated with the specified object version.

is_no_*relationship* **()**

Queries the database for every object that is not the target of a relationship of that name to any object.

For example, `is_no_successor` returns any objects that are not successors.

is_predecessor_of (’*object_name***’)**

Queries the database for all object versions that are immediate predecessors of the specified object version.

is_*relationship*_**of (’***objectspec***’, ’***operator***’,** *time***)**

Queries for all objects having the specified relation from the specified object whose relation create time matches the specified operator (that is, =, !=, >,<=, >, or >=) and time value.

For example:

```
is_associated_cv_of ( ’task23-1:task:M’, ’>’, time (’May 1, 2002’)
)
```

This query finds all the associated objects of task M#23 that were related to the task after May 1, 2002.

Note that `has_relationship(’objectspec’, ’operator’, time)` is the inverse query of `is_relationship_of(’objectspec’, ’operator’, time)`.

For information on how to create a relationship, see history command.

is_type_of (’*object_name***’)**

Queries for the *`type`* object version in the model that was used to create *`object_name`*.

recursive_is_member_of (’*project_name***’,** *order_spec***)**

Queries the database for all members of all projects of the project hierarchy specified by *`project_name`*.

The query returns a list of object version names, with all non-projects occurring first in the list followed by the projects that are members of the hierarchy. Only the

projects' positions within the results are significant to the search order. Use of other queries in conjunction with this query may change the result's order.

Note that the *project_name* **is not** returned from this query.

For a description of *order_spec*, see order_spec.

task ('*task_id***')**

Queries the database for tasks requests that have the specified number.

versions_in_a_baseline ('*project_spec***')**

Queries for project versions in a baseline. This query function expands to `"cvtype='project'` and `name='<project_name>'` and `instance='<project_subsystem>'` and `not is_no_project_in_baseline()'"`, which returns, for the project object, the set of all project versions that are in any baseline.

Relative operators

Relative operators are allowed in queries. The following table shows the relative operators available for constructing a query clause.

Operator	Description
`=`	The value of the attribute must be equal to the value of the constant.
`!=`	The value of the attribute must not be equal to the value of the constant; however, the attribute must exist.
`<`	The value of the attribute must be less than the value of the constant.
`<=`	The value of the attribute must be less than or equal to the value of the constant.
`>`	The value of the attribute must be greater than the value of the constant.
`>=`	The value of the attribute must be greater than or equal to the value of the constant.

Operator	Description
`match`	The value of the attribute must match a value of the constant, where the constant can include the wild card characters "`*`" and "`?`". The wild card "`*`" matches any number of missing characters and the wild card "`?`" matches a single missing character. The `match` operator is case-sensitive. You can use the wild card match only on the first 63 characters of an attribute.
`!match`	The value of the attribute cannot match any of the possible values of the constant.
`smatch`	The value of the attribute must match a string. The `smatch` operator is case-sensitive.

Logical operators

The logical operators you can use to construct query clauses are **`and`**, **`or`**, and **`not`**.

query_clause1 **and** *query_clause2*:

Only object versions that meet the requirements of both *`query_clauses`* are selected.

query_clause1 **or** *query_clause2*:

Object versions that meet the requirements of either *`query_clause`* are selected.

not *query_clause*:

Only object versions that do not meet the requirements of the *`query_clause`* are selected.

The `not` operator takes precedence over the `and` operator. The `and` operator takes precedence over the `or` operator. Use parentheses in a query to override the existing rules of precedence.

Constants

The constants allowed in query clauses are shown in the following table. Note that the class of attribute types found is determined by the constant specified. For example, if the constant is a string, only string attributes, or attributes that are a subtype of string, are found.

Constant type	Comments
`string`	Enclose strings in single quotes.
`integer`	Integer values can be zero to +-2147483647.
`boolean`	Boolean values can be `TRUE` or `FALSE`.
`time`	Time values must be in the form `time('time_string')`. See Date formats for more information.

Grouping Query Clauses

You also can group query clauses (without a depth limit) using parentheses.

Sample queries

- Find and change all `comment` attributes in the *working* object versions of the `proj_ico-1` project.

  ```
  ccm query "status='working' and is_member_of('proj_ico-1')"
  ccm attr -m comment -v "increase surface complexity" @
  ```

- Find and show all `executable`-type object versions associated with task 374.

  ```
  ccm query "type='executable' and is_associated_object_of('task374-
  1:task:probtrac')"
  ```

- Find all objects that are members of projects that have a) "web" in their names, and b) release values set to 6.0. (Use a nested query).

  ```
  ccm query "is_member_of(release='6.0' and name match '*web*')"
  ```

- Find all objects that are members of projects with a) the name Advanced_Topics, and b) release values set to 6.0. (Use a nested query).

  ```
  ccm query "is_member_of(release='6.0' and "name='Advanced_Topics'")"
  ```

- Find all members of project `toolkit-1.0`.

  ```
  ccm query "is_member_of('1/project/toolkit/1.0')"
  ```

- Find all members of the `editor-fcheng` project hierarchy using the fastest search method (by specifying "none").

  ```
  ccm query "recursive_is_member_of('1/project/editor/fcheng',none)"
  ```

- Find the cvtype in the model that was used to create the `csrc` object, `save.c-2`.

  ```
  ccm query "is_cvtype_of('save.c-2:csrc:1')"
  ```

- Find all objects owned by `linda` and modified with the past two days.

  ```
  ccm query -o linda "modify_time>time('-2:0:0:0')"
  ```

Relationships

A relationship enables Rational Synergy to relate one object to another object within a database. Relationships are maintained between objects (for example, a `csrc` object is a successor of the previous version it was checked out from), objects and tasks (for example, a task is associated with objects), change requests and tasks (for example, a task fixes a change request), and tasks (for example, a task fixes a task).

Note that the term *object* refers not only to versioned objects, but to any object that is stored in a Rational Synergy database, for example, projects, folders, tasks, change requests, etc.

The following topics explain Rational Synergy relationships.

- About relationships
- User-defined relationships
- Predefined relationships
- Query for related objects

About relationships

Each relationship has a name; its name identifies the nature of the relationship. Examples of relationship names include `successor`, `duplicate`, or `associated_task`. Relationship names can be any combination of alphanumeric characters.

A relationship is unidirectional. This means that it points **from** one object **to** the other object. For example, if `cosine.c-2` is the successor of `cosine.c-1`, the relationship points from version 1 to version 2. You can access the relationship from either object. For example, you can find that version 1 has successor version 2, and that version 2 is a successor of version 1.

An object can have any number of relationships. It can have many instances of the same relationship (with the same name) to different objects; for example, a single can have many associated tasks. An object can be on either end of a relationship; for example, an object can both have a successor object and be a successor to another object. An object can have many different types of relationships; for example, a task can be in a folder (`task_in_folder`), associated to source objects (`associated_cv`), and related to another task that fixes it (`fix`).

When you delete an object that has relationships, Rational Synergy automatically deletes the relationships too.

User-defined relationships

Rational Synergy provides interfaces for maintaining all predefined relationships; however, if you need new types of relationships, you can use the relate command to create them and the unrelate command to break them.

Teams might require new types of relationships for different reasons. You would need a new type of relationship if you wanted to be able to trace relationships between controlled objects that are useful to your team. For example, you might want to create a new relationship called `associated_spec` between a specification and the corresponding source code, to trace that relationship.

To relate or unrelate two objects from the CLI, you must specify the relationship name, the "from" object, and the "to" object. You can show relationships in a variety of ways, as described in the following examples:

- Show all relationships between two objects by specifying the "from" and "to" objects.
- Show all the other objects that an object is related to by specifying the "from" object.
- Show all the other objects that are related to an object by specifying the "to" object.
- Show all other objects that an object is related to through a specific relationship by specifying the "from" object and the relationship name.

By default, there are no rules to govern the security or semantics of user-defined relationships. For example, by default you can create a relationship between any two objects, whether or not they are modifiable. If you wanted a relationship to conform to semantics, such as one-to-many or type `problem` to type `task`, there is no way to define that in the relationship. Instead, you would need to write scripts or programs to maintain the relationship and enforce those rules, or you would need to customize Rational Synergy.

Predefined relationships

Rational Synergy provides interfaces for maintaining all predefined relationships. Use the Tasks tab to create, remove, or display `fix` relationships between two tasks; the History dialog to create, remove, or display `successor` relationships (links) between objects; and the Properties tab to create, remove, or display `associated_task` and `associated_cv` relationships (associations) between change requests and tasks or tasks and objects.

For best results, use the dialogs and commands provided to maintain the predefined relationships; use the relate command and unrelate command to support the user-defined relationships only.

The following table shows the predefined relationships shipped with Rational Synergy.

Name	Description
added_task_in_pg	From a project grouping to a task
associated_cv	From a task to a source object (many-to-many)
automatic_task_in_pg	From a project grouping to a task (many-to-many)
baseline_in_pg	From a project grouping to a baseline
baseline_project	From a project to its baseline project (many-to-one)
fix	From a fix task to a task (many-to-one)
folder_in_pg	From the project grouping to a folder
folder_in_rp	From the project to the folder (many-to-many)
folder_in_rpt	From the process rule to the folder (many-to-many)
folder_template	From the folder to the folder template (many-to-one)
folder_template_in_rpt	From the process rule to the folder template (many-to-many)
generic_pr	From release-specific process rules to generic process rules (many-to-one)
generic_pr_in_process	From processdef to generic process rules (many-to-many)
pr_in_release	From releasedef to process rule (many-to-many)
project_in_pg	From a project grouping to a project
reconfigure_template	From the process rule to the project (one-to-many)
removed_task_in_pg	From a project grouping to a task
saved_task_in_pg	From a project grouping to a task
successor	From a versioned object to its history successor (many-to-many)
task_in_folder	From a folder to a task (many-to-many)
task_in_rp	From the project to the task (many-to-many)

Query for related objects

Another way to show related objects is by using a query. `query` provides two query functions for relationships, one to query from each direction. The following shows the query syntax:

`is_`*`relationship-name`*`_of(`*`objectname`*`)`

`has_`*`relationship-name`*`(`*`objectname`*`)`

For example, to find `cosine.c-1`'s successors, you would use the following command:

```
ccm query "is_successor_of('cosine.c-1:csrc:1')"
```

To find the objects that have `cosine.c-2` as a successor, you would use the following command:

```
ccm query "has_successor('cosine.c-2:csrc:1')"
```

These functions also work for user-defined relationships; for example, to find the objects that are associated with `cosine.c-2`, you would use the following command:

```
ccm query "is_associated_spec_of('cosine.c-2:csrc:1')"
```

For more information about constructing queries, see Predefined relationships and Query clause elements

Shared projects

A *shared project* is a project in the *shared* state with members in the *visible*, *shared*, or any static state. A project in the *shared* state allows all users to make changes in the project. This behavior is unlike *working* or *prep* projects, in which only a single user or users in a particular role are allowed to make changes in the project.

The following topics introduce the concepts and uses of shared projects.

- How to determine whether shared projects will work for you
- Shared project methodology
- Parallel development and shared projects
- Troubleshooting shared projects

For more information about the dialogs and commands referenced in this document, see Rational Synergy *Help*.

How to determine whether shared projects will work for you

Shared projects have several advantages, mostly related to enabling multiple users to share a single project.

However, the same features that provide these benefits also impose limitations of which you should be aware. You should understand the limitations, as well as the benefits, so that you can determine whether shared projects are appropriate for your team.

- Benefits of shared projects
- Limitations of shared projects
- Best-case scenarios for using shared projects

Benefits of shared projects

Using shared projects has the following benefits.

- Time Savings

  You need not check out a *working* version of a large project (or multiple projects) merely to make minor changes. Unless you are maintaining a *working* version of a shared project, you need not perform maintenance or administrative operations; instead, you have immediate access to the project and its objects.

- Less Training

  Because all users can share the same project, individual users need not know how to administer the project (create, update, sync, set attributes, etc.). One user, such as the build manager, can be responsible for administering the project.

- Shared development capability

  Other users' changes are available immediately. For example, if Bob and Sue are working in the same shared project, Bob's changes are available to Sue as soon as he saves his changes.

- Less disk space needed

  Because all users are using a single project, multiple copies of the project's work areas are not needed. This feature can save disk space.

  Consider, however, that the amount of disk space saved is minimal if you use link-based work areas (with UNIX) or your team has few uncontrolled files in the work area.

Limitations of shared projects

Using shared projects imposes the following limitations.

- No insulation

  This is the most significant limitation of shared projects, because several situations can occur in which you might get unexpected behavior, particularly when you build products.

  Building in a shared project can have the effect of building in parallel, because users can build shared products at any time, and the new products are configured into the shared project automatically. Consequently, users' build results can change without notice. For example, if one user changes a library in a shared project, the change is included immediately, and can affect other users' subsequent builds.

  Incomplete changes to shared files also can cause unpredictable results, because the files are members of the project while they are being worked on, even if they have not been debugged.

  Finally, uncontrolled files in the shared work area can affect build results. For example, uncontrolled relocatable object files might be accessed by different users simultaneously, or in an order that corrupts the files or changes the build results.

  Remember that **the same feature that gives you immediate access to changes also delivers other users' incompatible changes, incomplete changes, or errors—immediately and without notice.**

  If you require an insulated project, you can check out and work in a *working* version of the project (see Parallel development and shared projects).

- Limited file system security

  Rational Synergy does not fully support file permissions for copy-based work areas of shared projects. All users have permissions to rename, move, or delete files in a copy-based work area, and they may be able to change files for objects checked out to other users.

In copy-based work areas, changes to work area files do not affect the controlled files in the Rational Synergy database. However, you should be careful to avoid reconciling unwanted changes from the work area back into the database.

Windows users can protect against unwanted changes to the work area by using non-shared drives for their work areas. (Note, however, that this method prevents multiple users from sharing the same work area.) UNIX users can protect against unwanted changes to the work area by making the their work areas link-based, because other users can change the links to the files, but only the files' owners can change the files themselves.

- Parallel development is prohibited

  Parallel development is prohibited in shared projects. After an object has been checked out from a shared project, no one else can check it out.

  You must not *use* a different version (`ccm use` command or Use dialog) and check out from that version, because you might remove another user's changes. Instead, you can check out a *working* project from the shared project and work on a parallel version (see Parallel development and shared projects).

- Common work area path required

  The work area of a shared project must be set to a path available to all users, and this path must be the same for all machines. Because all users must access the same work area from many different machines, the additional network traffic can slow performance.

Best-case scenarios for using shared projects

Shared projects are not suitable for all types of development projects; development projects suitable for shared projects typically have the project and development environment characteristics shown below. All aspects of the design should be considered when deciding whether to use shared projects.

Project characteristics

- Large projects with separate directories for each user's area of responsibility

  If your projects conform to this structure, you have the benefits of shared development, but you still reduce the risks inherent in the lack of insulation.

- Projects that do not build products

  Examples of projects that do not build products are projects that manage documents or Web pages, or manage integrations such as PowerBuilder.

  If a product must be built, you should modularize the build process (that is, create separate directories to provide some insulation). You might need to change makefiles to ensure that products are removed before being built; otherwise, file permissions might prevent one user from updating a file created by another user.

- Projects that do not require work areas on a local machine

  Because all users must be able to access the same work area, the project should be located on a shared drive, not on a local hard drive. Users might need to work on a local machine if they work disconnected from the data (for example, if they work on files at home or move the files to a local disk for better performance).

Development environment characteristics

- A small team that works well together and understands the issues caused by having uninsulated work areas
- Teams with users who are not trained to use all Rational Synergy features

  Users can perform basic Rational Synergy operations, such as check in and check out, without much knowledge of the product. However, some knowledge of how to create and manage projects is required.
- **All team members must understand the significance of uninsulated work areas.**

Shared project methodology

The methodology described in this section is task-based. Shared projects and standard projects have different lifecycles, and several CM operations behave differently in shared projects than they do in standard projects.

- Lifecycles
- Creating a shared project
- Working in a shared project
- Updating a shared project

Lifecycles

Two states are used only in shared projects: *shared* and *visible*. Where these states are used depends on the type of object.

Projects

The *shared* state is used for shared projects. The *shared* state is between the *working* and *prep* states in a project's lifecycle and allows all users to work in shared projects. By default, *working* objects are not used in shared projects.

Objects

The *visible* and *shared* states are used for checked-out objects in shared projects.

The *visible* state is used for shared development of regular (non-product) objects. Only the object's owner can change the object, but all users can use it (add it to their *working* projects or to a shared project).

The *shared* state is used for shared development of product objects. All users can change and use a shared object.

For complete information about lifecycles, see the *Introduction to Rational Synergy,* which fully describes project, object, and product lifecycles.This book is available at http://www.ibm.com/software/rational/support/.

Creating a shared project

Any user can create shared projects, but for best results, consider making the build manager responsible for shared projects, mainly because the build manager is familiar with project administration (creating, setting update properties, performing updates, etc.). However, in an environment where all users are familiar with projects, team leaders also can be responsible for shared projects. Making individual developers responsible for shared projects might also work well in a very small team in which the developers work together closely.

Using copy project

Although the project lifecycle starts at the *working* state, you can copy a project directly to the *shared* state by setting the project purpose to **Shared Development** in the Copy Project dialog. To create a shared project from the command line, use the `ccm copy_project` command with the `-purpose Shared Development` option.

From Rational Synergy, you can change the purpose of a project to **Shared Development**, which also changes the state.

Using create

Use the Create Project operation or `ccm create` command to create a project in the *working* state, then check in the project to the *shared* state. Be sure to set the next state to *shared*, because the next default state usually is *checkpoint* or *prep,* depending on your role.

Working in a shared project

When you work in shared projects, you must perform check-out and check-in operations differently than in standard projects.

Check out state

Regular objects are checked out to the *visible* state in shared projects. Product objects are checked out to the *shared* state in shared projects. You can check out subprojects manually to the *shared* state.

Note Always check out the object you want to change instead of making it writable by changing its file permissions. Changing only checked-out files helps ensure data integrity.

Automatic directory check-in

In shared projects, all commands that check out a directory automatically also check in the directory automatically. This features makes the shared directory available to all users so that they need not wait for a change to a directory member to be completed before making their own changes to the directory.

For example, if a user creates a new file in the `web` directory, the directory is automatically checked out as a result of the `create` command. Because the directory is checked in automatically, other developers can make changes while the user is changing his file. If the directory had not been checked in automatically, as well, the directory probably would have remained checked out (and unchangeable) until the user completed his changes.

Automatic directory check-in also requires that you set a current task before working in task-based shared projects. Without a current (default) task set, the directory is checked in without an associated task, and your change therefore cannot be selected by other projects during an update. Also, if a task is required on check-in and you have set no current task, the automatic directory check fails.

You can set the automatic directory check-in feature off by entering the following line in the `ccm.ini` file:

```
shared_project_directory_checkin = FALSE
```

No parallel development

You cannot check out from a checked-out, visible object; however, in a shared project you can replace a checked-out object by *using* a different version of the object (`ccm use` command or Use dialog). If you do use a different version of a checked-out object, you are *unusing* someone's checked-out object.

Again, edit only files checked out to you. Avoid changing files' permissions to make them writable.

Caution Do not cut another user's checked-out object. **Data loss can result from such an operation.**

Updating a shared project

When you work in shared projects, you must perform project update operations differently than in standard projects.

Update

If only one version of a shared project exists (no user has a *working* version of the project), a shared project requires almost no maintenance— the shared project is always up-to-date and updates are not required, provided that objects in the shared project have not been checked out in other projects. You can, therefore, create the shared project, then allow users to update the project using basic Rational Synergy commands (set the current task, check out, create, edit, and complete (check in) a task).

If *working* versions of a shared project exist, the build manager or team leader must update the shared project to gather the changes made in the *working* projects. The update properties should contain one folder, set up to query for all tasks for the current release. The folder should be writable by all and usable by all.

Updates can cause many objects to be updated, which affects the developers using the shared project. To lessen disruptions, perform updates when few users are using the project. Also, following an update, the build manager should review the update log for unexpected results and for parallel versions that must be merged.

Use the following process to provide all users with an up-to-date project:

1. The user responsible for managing the project (often the build manager) asks all users to check in their objects by completing their tasks.
2. Users complete their tasks.
3. The build manager updates the project.

Reconcile

All users can perform the reconcile operation in shared projects; however, each user can discard only object changes that are in a static state or checked out to him. Objects checked out to another user cannot be changed using the reconcile operation.

Use the `ccm resync` command to reconcile a single object.

Parallel development and shared projects

The following types of parallel development are common when using shared projects.

- Concurrent parallel development

  Parallel versions of an object exist, because two or more developers must change the same object at the same time (a single object checked out to multiple users).

- Parallel development for variants

  Multiple versions of the project exist, because two or more releases or platforms are required.

Concurrent parallel development

Developers can check out a *working* version of a shared project to perform parallel development or to work in an insulated version of the project. A *working* version of the project provides the insulation that a developer needs, and the *shared* version allows other developers to continue their work on the project.

For example, if you are working in a shared project and must make changes that will take several days to complete, you want to prevent changes made by other developers from interfering with your work. Instead of working in the shared project, check out a *working* version of the shared project. The project check-out operation will create a *working* project with the same members as the original project. The newly checked-out *working* project will contain *visible* and *shared* objects. To maintain your insulated area, you must remove the modifiable objects from your *working* project.

To check out a working version of a shared project, follow these steps:

1. Check out the project with an insulated development or collaborative development purpose.
2. Update your project.

These steps ensure that your *working* project is insulated from changes. When you update, the *visible* and *shared* objects will no longer be in your project. If you need additional information on checking out a *working* project, see the Rational Synergy help.

After you complete all the changes in a *working* project, the build manager can update the shared project to bring in the changes.

Parallel development for different platforms or releases

You might have several parallel versions of a shared project in different states, and these versions might support multiple platforms or releases.

For example, the shared project `airplane-hp_2.0s` (release 2.0 of airplane on the HP platform) might have parallel projects `airplane-hp_3.0s` (a future release), `airplane-hp_1.0p1` (a patch to a previous release), and `airplane-aix_2.0s` (a different platform for the current release). Version `hp_2.0s` is shared, but the other parallel projects need not be shared. In this example, `airplane-hp_1.0p1` should be in the *prep* state with a list of patches to be delivered.

Hierarchies of shared projects are maintained using the `platform` and `release` attributes. Continuing from the previous example, assume the airplane project consists of the subprojects `wing`, `fuselage`, and `rudder`. When `airplane-hp_2.0s` is updated, `wing-hp_2.0s` should be selected, not `wing-hp_3.0s`. The `release` attribute identifies matching versions of each subproject.

However, if you have multiple shared project hierarchies for the same platform and release, the update operation could select the wrong subproject version. Avoid this behavior by using a unique project purpose to check out hierarchies that have the same platform and release values.

For example, the following project purpose list supports three separate hierarchies of shared projects for the `airplane` project: a general purpose shared project, a shared project for a team of structural engineers, and a shared project for electrical engineers.

```
Shared Development: shared:
Shared - structural: shared: structural
Shared - electrical: shared: electrical
```

Instead of creating versions of the `airplane` project for each of three teams, the three teams work in three separate shared projects. All three project hierarchies can be updated regularly to keep them synchronized, or each team can determine how often to bring in changes from the other teams.

Troubleshooting shared projects

If you have problems using shared projects, consider looking for the following problems:

Directories containing incomplete changes

The automatic directory check-in feature can cause directories to contain incomplete changes if the directory is checked out a second time before the first change is finished. For example, Bob is assigned a task to remove the file `foo.c` in the `src` directory. He deletes `foo.c`, which checks out and immediately checks in the `src` directory. He then removes references to `foo.c` in other objects. Before Bob completes the task of removing the references, Sue moves a file from the `src` directory to another directory. The `src` directory is checked out and immediately checked back in. Sue completes the task. The build manager updates the *prep* project, which uses both Sue's and Bob's changes, even though Bob has not completed his changes.

Incomplete changes appear as configuration conflicts in the *prep* project. The problem can be avoided if the build managers show conflicts after updating *prep* projects.

Excessive busy state

In a shared project, Rational Synergy updates all dialogs affected by a change immediately following the change. This behavior can cause your interface to be busy while another user's operation is in progress.

For example, when Bob checks out an object that is in Sue's Project Explorer, Sue sees the busy cursor momentarily while the object is being updated; the busy cursor is usually not noticeable. This behavior prevents any changes to the object while it is being updated. However, if a user performs an operation that changes numerous objects (for example, an update), the cumulative effect of the changes can result in a persistent busy cursor, which flickers on and off, making it difficult to use any dialogs.

Because user's interfaces can be rendered temporarily unusable during change-intensive operations, you should avoid performing such operations during peak hours.

SOAD scopes

A *scope* is a modified query used by the Save Offline and Delete (SOAD) tool to create a list of objects to save offline and delete. The following advanced topics will help you understand how SOAD uses the scope to create a valid list of objects.

- Scope evaluation
- Scope validation
- Predefined scopes

Scope evaluation

Each Save Offline and Delete scope is evaluated as follows:

1. Any specified object is included in the initial object list. If the specified object does not exist, the evaluation quits with an error message.
2. Any query expression is evaluated (with any scope-specific exclusion query included), and the objects found are added to the object list.
3. If the object list is empty, the evaluation quits with an error message.
4. For each object in the query list, each applicable expansion rule is applied. If an expansion rule is type-specific, that rule is executed only if the object being expanded has that type. To avoid infinite recursion or duplicate expansions, the expansion keeps track of which objects have been expanded. When a query-based expansion rule is executed, any scope-specific exclusion query is added to the query expression. (This does not apply to ACcent-based expansion rules.)
5. For each object in the object list, any object whose type is one of the global excluded types (such as a model object) is removed from the list.
6. For each object remaining in the object list, if the objects are being saved offline any object that does not have other versions of the same instance is removed from the list.
7. The scope-specific exclusion rules are applied by evaluating each rule, in order, against the objects remaining in the list that match that type. If any of the objects returned by the query are **not** in the object list, the object being evaluated is removed from the list.
8. For each object remaining in the list, a query for projects using that object is performed. If an object has a parent project that is not on the object deletion list, it is removed.
9. Finally, if the user is not in the *ccm_admin* role, any object that is not modifiable by the user is removed from the list. This will allow non-administrators to use SOAD for deleting their own *working*, *checkpoint*, *visible*, *public*, or *prep* versions.

At the end of this evaluation, the object list contains all the objects targeted for deletion and optional saving offline.

You can select the Verbose check box in the Save Offline and Delete dialog box in Synergy Classic or use the `-verbose` option on the command line to obtain detailed scope evaluation information. However, keep in mind that turning on this type of tracing will slow the processing.

The following are scope evaluation features that change the object list after the initial object specification or query is evaluated:

- Global exclusions
- Keywords
- Expansion and exclusion rules

Global exclusions

If you were to use only a general, query-based method to delete objects, you could delete objects unintentionally that you need to preserve. SOAD therefore excludes the following types of objects:

- Model object types
- Candidate objects in excluded projects
- Last static versions of objects

Model object types

To prevent deleting objects that are required for operating the Rational Synergy database, SOAD excludes the object types listed in a model attribute named `soadf_excluded_types`.

The shipped set of excluded types is as follows:

```
model
mcomp
cvtype
attype
bstype
admin
tset
update_temp
folder_temp
```

Candidate objects in excluded projects

SOAD security rules prohibit deleting an object that is used in a project that is not being deleted. Because this is a global restriction, the rule is applied to all scopes.

Last static versions of objects

SOAD excludes any object that is the last (static) version of that object instance. Without such a restriction, after the object is deleted a user might create a new object with the same name and type, and the instance value would be reused. Restoring the original saved object would then be impossible because a `cluster_id` conflict would occur.

SOAD applies this rule by determining the depth of each version from the history root and choosing the static object with the highest depth. If the object has no static version, SOAD finds a non-static version with the highest depth, searching for a *prep* version first, then a version in any other state.

> **Note** This option is applicable only if you are also using the **Save Offline** option.

Keywords

You can use keywords in object names, queries, expansion and exclusion queries, and package names.

Keywords start with a "`%`" and are expanded as shown in the following table.

> **Note** You can define a literal string with a "`%`" character by specifying the character twice; for example, to obtain a string that expands to "`%release`," specify "`%%release`."

Keyword	Expanded in rules	Expanded in object name, queries and package name	Replaced with
`%1`	No	Yes	Value of first parameter
`%2`	No	Yes	Value of second parameter
`%3`	No	Yes	Value of third parameter
`%4`	No	Yes	Value of fourth parameter
`%5`	No	Yes	Value of fifth parameter
`%user`	Yes	Yes	User name
`%date`	Yes	Yes	Current date and time
`%object`	Yes	No	Name of object being processed

Expansion and exclusion rules

Expansion rules add objects to the object list based on their relationship to objects initially in the list. Exclusion rules remove objects from the object list based on their relationship to objects initially in the list.

SOAD provides predefined sets of expansion and exclusion rules that should be powerful enough to provide the expansion and exclusion behavior you need. You cannot use the GUI or CLI to change or add expansion or exclusion rules; however, you can edit the rules using a standard text editor.

Expansion rules are stored in a model attribute named `soadf_expansion_rules`. Exclusion rules are stored in a model attribute named `soadf_exclusion_rules`. Each entry is stored on a separate line with the query name, object type, query type, and values separated by '`:`'.

The predefined expansion rules are shown in the following table.

Expansion rule	Object type	Rule type	Description
Project's folders	project	query	Includes all folders used in the specified project's update properties.
Project's tasks	project	query	Includes all tasks used in the specified project's update properties.
Project's non-automatic tasks	project	query	Includes all non-automatic tasks used in the specified project's update properties.
Project's baseline project	project	query	Includes all baseline projects for the specified project.
Project's members	project	query	Includes all objects that are direct members of the specified project.
Project's recursive members	project	query	Includes all objects that are direct or recursive members of the specified project.
Project's products	project	query	Includes all products that are direct members of the specified project.
Hierarchy project members	project	query	Includes all objects and projects that are direct or recursive members of the specified project.

(Continued)

Projects using baseline project	project	query	Includes all projects that are baselines of the specified project.
Non-static projects using baseline project	project	query	Includes all non-static projects that are baselines of the specified project.
Project's baselines	project	query	Includes all projects that have the specified project in their baselines.
Folder's tasks	folder	query	Includes all tasks that are in the specified folder.
Folder's non-automatic tasks	folder	query	Includes all non-automatic tasks that are in the specified folder.
Projects using folder	folder	query	Includes all projects that use the specified folder in their update properties.
Task's objects	task	query	Includes all objects that are associated with the specified task.
Task's baseline	task	query	Includes all baselines that use the specified task.
Projects using task	task	query	Includes all projects that use the specified task in their update properties.
Folders using task	task	query	Includes all folders that use the specified task.
Baseline's projects	baseline	accent	Includes all projects that are in the specified baseline, and did not exist before the baseline was created.
Baseline's tasks	baseline	query	Includes all tasks that are in the specified baseline.
Tasks associated with object		query	Includes all tasks that have the specified associated object.
Project's project grouping	project	query	Includes all project groupings that contain the specified project.

The predefined exclusion rules are shown in the following table.

Exclusion Rule	Object Type	Query Type	Description
Baselines used by other project groupings		query	Excludes all baselines in use by other project groupings
Baseline projects used by other projects	project	query	Excludes all baseline projects in use by other projects.
Baseline projects used by other non-static projects	project	query	Excludes all baseline projects in use by other non-static projects.
Projects used by other baselines	project	query	Excludes all projects in use by other baselines.
Projects that are the last static version	project	accent	Excludes all projects that are the last static versions of the included projects.
Folder used by other projects	folder	query	Excludes all folders used in other projects' in update properties.
Task used by other projects	task	query	Excludes all tasks used in other projects' in update properties.
Task used by other folders	task	query	Excludes all tasks in use by other folders.
Task used by other baselines	task	query	Excludes all tasks in use by other baselines.
Objects associated with other tasks		query	Excludes all objects associated with other tasks.
Objects associated with other non-automatic tasks		query	Excludes all objects associated with other non-automatic tasks.
Attachments of any change request		query	Excludes all attachments associated with change requests.
Objects that are the last static version		accent	Excludes all objects that are the last static versions of the included objects.
Project groupings containing other projects	project_ grouping	query	Excludes project groupings containing other projects than the one being deleted.

Scope validation

SOAD validates scopes as follows:

1. Before saving a new scope.
2. Before saving an edited scope.
3. Before scope evaluation.

The validation ensures the following:

- Referenced expansion rules exist.
- Referenced exclusion rules exist.
- The scope query syntax is valid.
- The scope exclusion query syntax is valid (although the query can still fail on evaluation).
- Keywords are valid in the scope object, scope query, scope exclusion query, and scope package name.

> **Note** The query syntax check only determines whether the syntax is valid: It does **not** check that query functions have valid names and valid arguments, or that attribute values are appropriate. Therefore, a query expression that passes a syntax check might still fail when evaluated.

Predefined scopes

Each predefined Save Offline and Delete scope is defined and stored in an XML file in the *CCM_HOME*`\etc\soad directory` (Windows) or `$CCM_HOME/etc/soad` directory (UNIX). The file name is the scope name, encoded as a URL, with spaces replaced by `%20` and an "`.xml`" suffix.

> **Note** Even though you can edit, create, and delete scope files using a text editor if you are logged in as *ccm_root*; it's best to use the GUI or CLI for these operations.

The following are two examples of predefined scopes:

- Release-based scope
- Project hierarchy-based scope

Release-based scope

The following is the content of the XML file for the predefined scope "`All projects and related objects for a specified release`":

```
<?xml version="1.0" encoding='ISO-8859-1'?>

<soadfscope version="1">
  <predefined>TRUE</predefined>
  <role></role>
  <parameter>
    <label>Release value</label>
  </parameter>
  <object></object>
  <query>release='%1' and cvtype!='problem'</query>
  <expansion_rule>Folder's tasks</expansion_rule>
  <expansion_rule>Project's folders</expansion_rule>
  <expansion_rule>Project's tasks</expansion_rule>
  <expansion_rule>Task's objects</expansion_rule>
  <exclusion_rule>Baseline projects used by other non-static projects
  </exclusion_rule>
  <exclusion_rule>Folders used by other projects</exclusion_rule>
  <exclusion_rule>Objects associated with other non-automatic tasks
  </exclusion_rule>
  <exclusion_rule>Projects used by other baselines</exclusion_rule>
  <exclusion_rule>Tasks used by other baselines</exclusion_rule>
  <exclusion_rule>Tasks used by other folders</exclusion_rule>
  <exclusion_rule>Tasks used by other projects</exclusion_rule>
  <exclusion_query></exclusion_query>
  <package_name>All projects and related objects for Release %1 saved
  on %date</package_name>
</soadfscope>
```

First, the initial object list is created by querying for all objects that have a specified release.

Next, for each project found, expansion rules add the following to the list:

- each project's folders and tasks
- each folder's tasks
- each task's associated objects

Finally, exclusion rules remove the following from the object list to prevent damage to folders, tasks, and baselines that are not being deleted:

- baseline projects used by other non-static projects
- projects used by other baselines
- folders and tasks used in other projects
- tasks used by other baselines or folders
- objects associated with other non-automatic tasks

Project hierarchy-based scope

The following is the content of the XML file for the predefined scope "`Project hierarchy and related folders and tasks`":

```
<?xml version="1.0" encoding='ISO-8859-1'?>

<soadfscope version="1">
  <predefined>TRUE</predefined>
  <role></role>
  <parameter>
    <label>Project objectname</label>
  </parameter>
  <object>%1</object>
  <query></query>
  <expansion_rule>Folder's non-automatic tasks</expansion_rule>
  <expansion_rule>Project's folders</expansion_rule>
  <expansion_rule>Project's non-automatic tasks</expansion_rule>
  <expansion_rule>Project's recursive members</expansion_rule>
  <exclusion_rule>Baseline projects used by other non-static projects
  </exclusion_rule>
  <exclusion_rule>Folders used by other projects</exclusion_rule>
  <exclusion_rule>Objects associated with other non-automatic tasks
  </exclusion_rule>
  <exclusion_rule>Projects used by other baselines</exclusion_rule>
  <exclusion_rule>Tasks used by other baselines</exclusion_rule>
  <exclusion_rule>Tasks used by other folders</exclusion_rule>
  <exclusion_rule>Tasks used by other projects</exclusion_rule>
  <exclusion_query></exclusion_query>
  <package_name>Project hierarchy %1 saved on %date</package_name>
</soadfscope>
```

First, the initial object list contains only the project specified by the object name.

Next, for the specified project, expansion rules recursively add the following to the list:

- the project's folders, non-automatic tasks, and recursive members
- each folder's non-automatic tasks
- each task's associated objects

Finally, exclusion rules remove the following from the object list to prevent damage to folders, tasks, and baselines that are not being deleted:

- baselines used by other non-static projects
- projects used by other baselines
- folders used by other projects
- tasks used by other baselines, projects, or folders
- objects associated with other non-automatic tasks

Triggers

Using notification triggers

The notify feature allows you to define programs that are to be called when objects change state, or when change requests are submitted. The programs may be written in any language (or may be shell scripts or batch files), and may be passed literal values and/or the values of attributes of the object on which the trigger was invoked.

The following topics are discussed:

- Format and description of trigger definition files
- Programs
- Messages
- Examples

Format and description of trigger definition files

Following is a sample of a trigger definition file, followed by descriptions of components of the file.

```
trans_type
{
   type
   type
     .
     .
     .
   {
      status  program  arg1 arg2 ... argN;
      status  program  arg1 arg2 ... argN;
              .
              .
              .
   }
   .
   .
   .
}
.
.
.
```

Component descriptions

`trans_type`

Denotes when the program is to be called:

`pretransition`

The program is called before any work takes place on the transition. This is applicable for validating conditions on non-PT objects before proceeding with the rest of the trigger. If the pretransition fails, the entire transaction will fail.

`posttransition`

The program is called after the transition has completed successfully. This is applicable for transaction on non-PT objects. The return status on posttransitions are ignored.

`pt_transition`

The program is called after a PT object has transitioned. The program is specially set up for the transition of PT-related objects.

`type`

Defines the type of an object. It should be a valid database type, or '`*`' to denote all types. If an all types transition is defined, it will be executed before the specific transitions are called.

`status`

Defines the status of an object. It must be a valid status (i.e. *working, integrate, test, sqa, released, public*, etc.).

`Arguments`

Arguments in double quotes are treated as literals, and are passed to the trigger program as literals, without the quotes.

Arguments that begin with an "`=`" symbol are treated as template files. The file is read into memory, keyword replacement is performed using the attribute values from the object, and the altered file in then written out to disk. The path of the file is then passed to the program. It is the responsibility of the program to remove the file after use.

The other arguments are assumed to be the names of attributes on the transitioning object. The value of the attribute will be passed as the argument to the notify program. If the name of the argument is preceded by '`+`', the trigger is not invoked if the attribute is missing or has an empty value. Be aware that some attributes are

created or set after an object has changed state, so at the instant a trigger is called, some attributes might not be present, or might not have the values you expect.

Multi-line text arguments (attributes and literals) cannot be passed. If the attribute '`source`' is specified, then the path to the source file or the cache file will be passed.

The keywords `dbid`, `database`, and `current_user` are substituted with the database ID, the absolute path name to the database, and the name of the user running the current CM session.

Programs

Programs are in the `notify` directory of the database containing the objects or the `CCM_HOME/bin/util` directory of the run area. For security reasons, you may not specify a relative or absolute path in the trigger definition file; you can write a wrapper program in the `notify` directory, and from there call programs not in the `notify` directory.

If the trigger is running on the interface (`Trig_ui.def`), the program is executed by the current user. If the trigger is running on the engine (`Trig_eng.def`), the program is executed by `ccm_root`. You should consider the security implications of this.

> **Note** CCM commands are not supported within the triggered programs. Executing a `ccm` command via the triggers will cause your session to hang.

Messages

The programs can return informational messages and instructions back to the Rational Synergy session through stdout, stderr, or the Exit Status code.

stdout

```
ATTR_SET:<name>:<type>:<value>
ATTR_ADD:<name>:<type>:<value>
```

Both will create the attribute of the specified name and type if it doesn't already exist. `ATTR_SET` overrides any existing value, whereas `ATTR_ADD` appends information to a text attribute.

```
MSG:<string>
```

Messages in <`string`> will be displayed to the user. If it was triggered from the GUI, the messages will be displayed in a dialog box. If it was triggered from the CLI, messages will be displayed to the command line.

APPLY_ATTRS_ON_FAIL

Generally, any attributes returned will not be applied if the program returns a failure message. If this string is returned, then the attributes will be applied whatever the return status.

IGNORE_FAIL

As mentioned above if any program fails in a group, then the whole transaction will fail. Returning this allows the program to fail, but for the transaction still to succeed. Note that the program is still considered to have failed; it just won't affect the overall result.

If this is set to true, then all other lines are treated as MSG.

FAILED

If this message is seen, the program is assumed to have failed, and the exit status is ignored.

<string>

All other output to stdout will be printed to the log files and message panel. This differs from MSG: in that MSG: output is displayed in a pop-up dialog if the trigger is invoked from the GUI.

stderr

All output to stderr is displayed in the log files and message panel, and in a pop-up dialog if the trigger is invoked from the GUI.

If several program are running for a transaction then all output will be concatenated and displayed as one message.

Exit Status

An exit status of zero (0) is considered success. If pretransition receives any other status, or if it receives the FAILED message, it will not allow the transition.

Examples

Although the triggers can be executed on UNIX or Windows, the examples below call Windows batch files to demonstrate the capabilities of triggers where the client is running under Windows.

```
#  Before ANY object gets checked in to integrate state,
#  run the pretrans.bat script, passing the owner, object name,
#  and object version
pretransition
    {
    *
        {
        integrate pretrans.bat owner name version;
        }
    }

#  After ascii and csrc objects are created, call the psttrans.bat script,
#  passing the owner, name and version.
posttransition
    {
    ascii
    csrc
        {
        working psttrans.bat owner name version;
        }
    }

#  After a problem has been verified, call the verified.bat script,
#  only if the submitter_email attribute is present and not empty.
#  Pass the script the name of a file that has the keyword expanded
#  contents of the verified.tpl template.
pt_transition
    {
    problem
        {
        verified verified.bat +submitter_email =verified.tpl;
        }
    }
```

```
#  Here is an example of the template you might use with the trigger
#  for email problem submissions, notifying the remote sender of the
#  new problem number.
#  The keywords you may use are any attributes of the object in question,
#  or one of the following special keywords:
#   %dbid           DCM database id
#   %database       Database path
#   %current_user   Current user name
#   %rfc822_date    Date in the RFC822 format Wed, 13 Oct 99 02:20:24 PDT
#  Attributes or keywords may be specified in the format %name or %{name}
To: %submitter_email
From: %current_user
Subject: Problem Receipt Notification
Date: %rfc822_date

Problem Receipt Notification
_____________________________________________________________________________

Your problem report:

%problem_synopsis

was received and created as number %problem_number
in database %{database} by %{enterer}.
```

Work area

A *work area* is a region in your file system into which Rational Synergy writes a project hierarchy. The project's file structure in the work area therefore maps to the project's database file structure.

In Windows, the work area contains copies of the database objects, which Rational Synergy keeps synchronized with the database.

In UNIX, a work area is either link-based (containing links to the database objects) or copy-based (containing copies of the database objects). In both types of work areas, Rational Synergy keeps the work area synchronized with the database.

The following topics explain how work areas function:

- How work areas are updated
- Work area locations
- Absolute and relative work areas
- Updating work area paths

For more information about the commands referenced in this document, see their Rational Synergy help topics.

How work areas are updated

Rational Synergy updates your work area automatically and transparently when you create or change a project: when you add members to a project, the work area is updated with the new files, and when you remove members from a project, the corresponding files are removed from your work area. You also can update work areas manually.

The following topics explain how Rational Synergy updates work areas:

- Updating copy-based work areas
- Updating link-based work areas
- Changing or recreating work areas
- Updating multiple work areas

Updating copy-based work areas

A **copy-based** work area contains a copy of the source for every object in your project. At least two copies of each file always exist: one in the work area and one in the database.

Whenever you access a controlled file, Rational Synergy checks the file to determine whether it has changed. If the object is checked out and you are making changes to the file in the work area, Rational Synergy automatically updates the database with the changes the next time the object is accessed. If you are making changes in your work area to a file that is not checked out, Rational Synergy notifies you that a conflict exists

when you access that file in the work area. See the reconcile command or the Sync Work Area operation to resolve work area conflicts.

All work areas in Windows are copy-based. To make a work area copy-based on UNIX, either create the project during a remote client session (using `ccm start -rc`), use the `ccm work_area -cb` option, or select the Symbolic Links or Copies option in the Properties dialog box.

Use copy-based work areas on UNIX if one of the following requirements is met:

- You do not have an NFS mount to the database from the client host.
- You need to work disconnected from the database, such as when you are working outside of the office.
- You are using a tool that does not use symbolic links properly.

Updating link-based work areas

By default, UNIX client users work from a **link-based** work area. The work area is called link-based because the files in your work area are symbolic links to the files in your Rational Synergy database. When you edit a file in a link-based work area, the database files are updated immediately with changes. This occurs because you are working on the files themselves, rather than on copies of them.

If you are running on the UNIX client, you can use either a copy-based or link-based work area. UNIX clients that start a remote client use copy-based work areas.

> **Note** If you use a tool or command in your work area that breaks symbolic links, you must sync your project to update the database objects with the work area changes.

Changing or recreating work areas

You can perform the following operations to examine or change a work area:

- See any work area and database conflicts (before performing a work area operation)

  If you would like to compare your controlled work area files with the database files, see the reconcile command.

- Change link-based to copy-based, and vice versa

  If you want to change a work area from link-based to copy-based, or vice versa, use the `ccm work_area -cb` option or select the Symbolic Links or Copies option in the Properties dialog box to convert the work area automatically. See the reconcile command for information on converting link-based or copy-based work areas.

- Delete

  You can delete a work area (for example, to remove unwanted, uncontrolled files). However, you should always sync before you delete a work area to ensure that the

database has been updated with your work area changes. See the reconcile command.

- Re-create

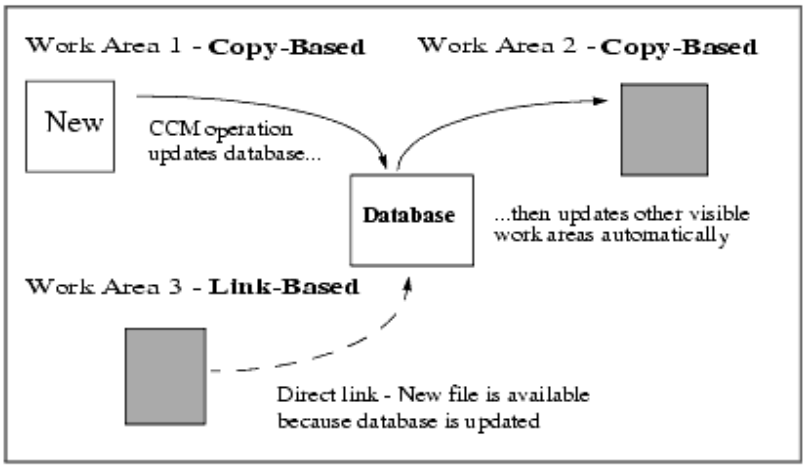
  If you removed files from your work area and you want to write the database versions back into the work area, you can re-create the work area using the sync command.

Updating multiple work areas

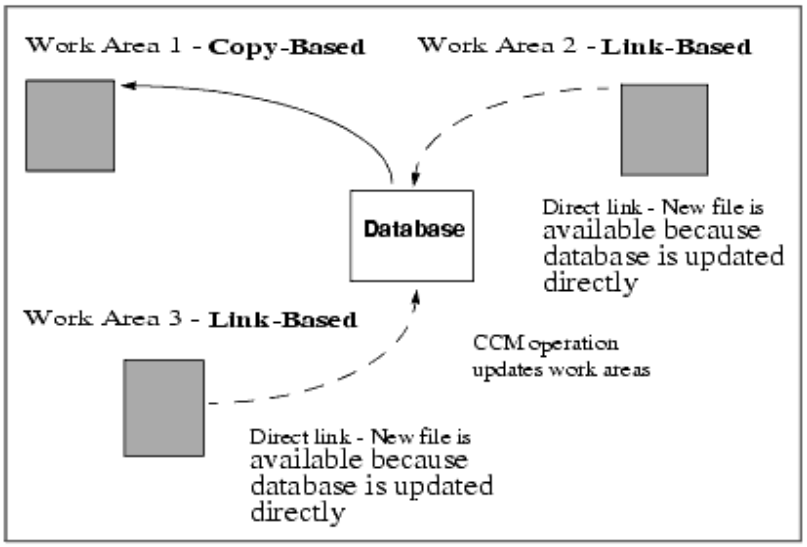
Objects such as header files and libraries can be used in multiple projects, and can therefore reside in more than one work area. Rational Synergy keeps the files synchronized in multiple locations by updating the files when they are accessed.

Recall that for each copy-based work area in which a file is used, there is a copy of the file. As you make changes to a file in a copy-based work area, you change only its local copy. The next time you access the file, Rational Synergy detects the change and updates the file in the database and in any other visible work areas in which the file is used.

The example in the figure below shows how updates occur when a copy-based file that is used in multiple projects is updated. The boxes represent copies of a single source file or links to the source file. Only visible work areas are updated.



The second example shows how updates occur when a link-based file used in multiple projects is updated. The boxes represent copies of a single source file or links to the source file. If a change is made through a link, copy-based work areas (such as Work Area 1, below) are not updated until the object is accessed through a client to which the work area is visible.



If you have a file that is used in multiple projects and you change that file in more than one work area at a time, you are notified that a conflict exists when you access the changed file through Rational Synergy. You must resolve the conflict before continuing.

Work area locations

Each project version has a **single** work area location. You can change this location, but the work area always is maintained in a single location.

- If you want to synchronize the same project to multiple locations, you must check out separate versions of the project for each work area.
- If you want to make this work area available to multiple users, sync the project to a location available to all users.

If you check out separate versions of a project, you will see all of them when you enter a command that lists your work area files. The following output shows a Windows work area, then a UNIX work area.

Windows:

```
Directory of C:\ccm_wa\ccmdb219
.              <DIR>     02/01/99  3:04p .
..             <DIR>     02/01/99  3:04p ..
demo-1         <DIR>     02/01/99  3:04p demo-1
demo-2         <DIR>     02/07/99 11:23p demo-2
demo-3         <DIR>     02/21/99  9:15a demo-3
```

UNIX:

```
drwxr-xr-x   4 bill      develop       2048 Aug 31 14:37 ./
drwxr-xr-x   3 bill      develop        512 Aug 31 14:37 ../
-r--r--r--   1 bill      develop         63 Aug 31 14:37 .ccmwaid.inf
drwxr-xr-x   2 bill      develop       2560 Aug 31 14:37 demo-1/
drwxr-xr-x   2 bill      develop       1024 Aug 31 14:37 demo-2/
drwxr-xr-x   2 bill      develop        512 Aug 31 14:37 demo-3/
```

To go to the correct project, you must change directories to the appropriate project **and** version (*project-version*). In the output shown above, there are three versions of the `demo` project: `demo-1`, `demo-2`, and `demo-3`.

One consequence of a single-location work area is that if a project is synchronized to a local Windows file system, UNIX sessions cannot see this work area, and vice versa. Therefore, use different versions of the project for different platforms.

Absolute and relative work areas

Rational Synergy supports two kinds of work areas for subprojects: absolute and relative. The following topics describe these types of work areas.

- Absolute work areas
- Relative work areas

Absolute work areas

Absolute work areas exist as separate directory hierarchies; that is, an absolute subproject's work area path need not be under its parent project's path.

You can sync the subproject anywhere; however, by default, the subproject resides under your home directory in the following path:

Windows: *home_directory*\ccm_wa*database_name**project-version*

UNIX: ~/ccm_wa/*database_name*/*project-version*

This means that other projects can find and use absolute subprojects even if the projects' work areas are not located in the same directory structure.

For example, suppose `bar-1` is a subproject of `foo-1` in the `ccm_tools` database and (UNIX only), you are not using symbolic links. If `bar-1` is absolute, the work area will appear as follows:

Windows:

```
c:\ccm_wa\ccm_tools
    foo-1\
        foo\
            a.c
            b.c

    bar-1\
        bar\
            c.c
```

UNIX:

```
/users/bill/ccm_wa/ccm_tools
    foo-1/
        foo/
            a.c
            b.c
    bar-1/
        bar/
            c.c
```

You can use an absolute project as a subproject more than once. Rational Synergy expects developers to put absolute projects meant for use by multiple developers on a shared file system. This is particularly useful for external projects, such as those that store shared products, libraries, and header files.

On UNIX, when you are using symbolic links, absolute subprojects appear as subdirectories in their parent projects' work areas. For example, if the subproject is absolute, you might see the following project hierarchy:

```
/users/bill/ccm_wa/ccm_tools
    foo-1/
        foo/
            a.c
            bar -> /users/bill/ccm_wa/ccm_tools/bar-1/bar
    bar-1/
        bar/
            c.c
```

On a Windows client when a subproject is absolute, if you look at the parent project's work area you will not see the subproject as a subdirectory because Windows does not support symbolic links.

Using a work_area command operation, you can change a subproject from absolute to relative.

Relative work areas

When used by another project, a **relative** subproject resides in the parent project's work area as if it were a subdirectory. This is useful when you must structure your code into subprojects for performance reasons, or if your makefiles or tools are written to use relative paths.

The following examples show the work area for the same projects shown in the previous example, except that the `bar-1` subproject is relative instead of absolute:

Windows:

```
c:\ccm_wa\ccm_tools
    foo-1\
        foo\
            a.c
            b.c
        bar\
            c.c
```

UNIX:

```
/users/bill/ccm_wa/ccm_tools
     foo-1\
         foo\
             a.c
             b.c
             bar\
                 c.c
```

You can use a relative subproject as a subproject in multiple projects as long as it is static (cannot be modified). A relative subproject that is modifiable can be used only once because it resides in its parent project's work area and can be synchronized only to one location. If you want to use a modifiable relative project in multiple locations, you must use multiple versions of the project.

Projects on both the Windows and UNIX clients are absolute by default when they are created. If you check out a new version of a project, the new version's work area is relative only if you check it out from a relative project. Otherwise, it is absolute.

If the makefiles in a project hierarchy reference the members of a subproject through a relative path as if the subproject is a subdirectory, or if you are unable to use a symbolic link to a subproject directory, you must keep the project relative. If the makefiles in a project hierarchy reference the members of a subproject as if the subproject is in an entirely unrelated directory structure, the subproject can be absolute.

You can change your makefiles to work with either absolute or relative projects, or a combination of both, or you can set up your subprojects as either absolute or relative so that your existing makefiles can recognize the work area directory structure.

Updating work area paths

If you want to synchronize a project to a different location in the file system, you must change its work area path. Also, if you copy or move a database, or change its version delimiter, you must change its path settings.

The following topics explain work area paths and some of the changes you might make to them.

- Elements of a work area path
- Moving a work area
- Changing a database
- Security and visibility issues
- work_area command syntax

Elements of a work area path

A typical project work area path consists of a user's home directory location, a work area subdirectory (for example, `ccm_wa`), the database name, the project's name and version, and the project's root directory.

The typical syntax is as follows:

```
Windows: home_dir\ccm_wa\database_name\project_name-version\project_name
UNIX:    home_dir/ccm_wa/database_name/project_name-version/project_name
```

For example, the following path is for user `bill`'s `baselib-bill` project in the `ccmdb219` database:

```
Windows:  c:\ccm_wa\ccmdb219\baselib-bill\baselib
UNIX:    /users/bill/ccm_wa/ccmdb219/baselib-bill/baselib
```

Moving a work area

To move a project's work area, use the `ccm work_area -setpath` command, or the Properties dialog box. Note that these interfaces append *project_name<version_delimiter>project_version* to the work area path automatically; you need not specify that part of the work area path.

> **Note:** UNIX link-based work areas represent a project's controlled objects using symbolic links to the corresponding files in the database path. If you move a database, you must update all symbolic links in the work areas to point to files located in the new database path.

Changing a database

You might need to update the work area path if you rename, move, or copy a database (for example, use the `ccmdb cp` or `ccmdb unpack` command), or if you change a database's

version delimiter. Use the `work_area` command's `-find` option to help identify and update those projects whose work area paths are out-of-date.

In addition, each work area contains an identification file, `ccmwaid.inf (Windows) or .ccmwaid.inf (UNIX)`, which contains the path name to the project's database. Rational Synergy uses this file to ensure that only one database updates a given work area. If you move a database, the work area identification file appears to be for a different database, making the work areas unusable. Use the `work_area` command's `-dbpath` option to help identify and update work area identification files to account for changes in database location.

Security and visibility issues

To change a work area path, you must have Rational Synergy write access to the project. You must be in the *ccm_admin* role to update static projects. File system write access to work areas also is required to update any project.

The new work area path must be visible to the Rational Synergy client. If work areas are located on both Windows and UNIX file systems, you must use separate Rational Synergy sessions to update those work areas. To update Windows work areas you must use a Windows client session to which the Windows file systems are visible; to update UNIX work areas you must use a UNIX client session to which the UNIX file systems are visible.

You must have write access to the work area in the file system to update a work area identification file.

work_area command syntax

The following examples show the syntax for the `-find` and `-dbpath` options only. For more information, see the work_area command.

- Replace *`find_str`* with *`new_str`* in the work area of all projects found (in the specified scope, or specified with the `-p` option).

  ```
  ccm work_area -find find_str -replace new_str
  ```

  If you also specify `-reg` or `-regexp`, both *`find_str`* and *`new_str`* are interpreted as regular expressions.

- Locate projects with work area identification files containing the database path name specified with the *`old_path`* argument, and update those files with the current database's path name.

  ```
  ccm work_area -dbpath old_path
  ```

  The *`old_path`* option cannot be interpreted as a regular expression. The `-dbpath` option updates the work area ID files for the found projects with the path name of the current database. This option should be used only when a database is moved. Use of this option on a UNIX link-based work area causes the work area to be synchronized in order to update the links to the new database location. If you also specify the `-`

`nosync` option, the synchronization is deferred; however, you must perform the synchronization manually before the work areas can be used.

You can add the `-new` option to `-find` and `-replace` to indicate the presence of a new database. This means that the original work areas specified by `-find` are not visible to this session and should be ignored. This option is useful after a database has been unpacked to a new path name and you want to ignore the original database work areas, or if the original database is not present. If you do not add the `-new` option, the command operates only on projects with visible work areas.

You can add the `-show` option to `-find` or to `-dbpath` to show which projects are updated. If you use `-show` with `-find` and `-replace`, the replacement path names are displayed. This can be very useful if you are using `-find` and `-replace` with regular expressions (`-regexp`).

Examples

- The following command finds all your *working* projects with visible work areas and paths that contain the `"-"` character, and changes `"-"` to a `"~"`:

  ```
  ccm work_area -find "-" -replace "~"
  ```

  You could use this command after changing a database delimiter from `"-"` to a `"~"`. You must execute the command from enough sessions to change all work areas. If all your work areas are visible from one session, one session is sufficient. However, if you have both Windows and UNIX work areas, you must execute this command from both Windows and UNIX clients.

- The following command finds all *prep* projects with visible work areas and paths that contain the `"-"` character, and changes `"-"` to a `"~"`:

  ```
  ccm work_area -find "-" -replace "~" -scope prep
  ```

  You could use this command after changing a database delimiter from `"-"` to a `"~"`. You must execute the command while working as a build manager from enough sessions to change all build management work areas. If all *prep* work areas are visible from one session, one session is sufficient. However, if you have both Windows and UNIX build management work areas, you must execute this command from both Windows and UNIX clients.

- On UNIX, the following command finds all of your *working* projects with work areas for the `/vol/acrel5/ccmdb/ccm_platform` database, and updates the work area ID files with the path to the current database location:

  ```
  ccm work_area -dbpath /vol/acrel5/ccmdb/ccm_platform
  ```

Use this command to update a database that has been moved but can use its old work area paths. You must execute the command from enough sessions to change all work areas. If all your work areas are visible from one session, one session is sufficient. However, if you have both Windows and UNIX work areas, you must execute this command from both Windows and UNIX clients.

- The following command finds all of your working projects with paths that contain the `platform` string, and changes the string to `services`:

  ```
  ccm work_area -find platform -replace services -new
  ```

  Use this command to update a database that was unpacked or copied to a new name. Note that the `-new` option creates all new work areas because the old work areas are in use by the old database. If you want to reuse the old work areas (such as for a database that has been moved), you first must update the work area ID files using the `-dbpath` option so that the old work areas are visible for this database.

Regular expression examples

Use the `-reg` or `-regexp` option to cause the `work_area` command to interpret the *`find_str`* and *`new_str`* arguments as regular expressions. For additional information, see Regular expressions.

Regular expressions can be useful, but have the following limitations:

- Windows clients use the backslash for directory names. A backslash in a regular expression might be interpreted as an escape or as part of a replacement construct.
- If you enclose a *`find_str`* or *`new_str`* argument in quotes, the command line processing can become confused because quotes are interpreted by both the UNIX shell the Rational Synergy command line processor (even on Windows clients).

The following examples show some work area path changes using regular expressions.

- Shorten all work area paths under `c:\ccm_wa\bill45\` to `c:\ccm_wa\`.

  ```
  ccm wa /find "bill45\\\\" /replace "" /reg /p
   Checking work area paths for replacement...
   1 project(s) will be checked.
   Setting path for work area of 'hsai~1' to 'c:\ccm_wa\hsai~1'...
   1 project work area path(s) were updated:
       'hsai~1': 'c:\ccm_wa\hsai~1'
  ```

  When you remove a directory from a path, include an associated backslash. Specifying the leading backslash is simpler, but the following example uses the trailing backslash:

  ```
  bill45\.
  ```

  The Rational Synergy command processor sees the leading quote for `bill45`, then if it encounters a backslash while looking for the trailing quote, it interprets the backslash as a signal to include the following quote as part of the argument instead of

as the closing quote. Therefore, you must "escape" the backslash by prepending another backslash. Also, the expression that results from the Rational Synergy command processing (`bill45\`) will be misinterpreted by the regular expression processor as a backslash without a corresponding replacement construct character, unless the trailing backslash is, itself, escaped with two additional backslash characters.

If you specify multiple directories enclosed in quotes, you must use only four quotes for the trailing directory backslash, immediately preceding the closing quote. The directory backslash in the middle of the argument can be escaped once so that the regular expression processor does not interpret the backslash as the beginning of a replacement construct.

```
ccm wa /find "ccm_wa\\bill45\\\\" /replace "" /reg /p hsaw~1
Checking work area paths for replacement...
 1 project(s) will be checked.
 Setting path for work area of 'hsaw~1' to 'c:\users\bill\hsaw~1'...
 1 project work area path(s) were updated:
     'hsaw~1': 'c:\users\bill\hsaw~1'
```

- Remove a directory with a leading special character from a path, using a trailing backslash, shortening a work area path from `c:\ccm_wa\+bill45\hsaw~1` to `c:\ccm_wa\hsaw~1`.

```
ccm wa /find "\+bill45\\\\" /replace "" /reg /p hsaw~1
 Checking work area paths for replacement...
 1 project(s) will be checked.
 Setting path for work area of 'hsaw~1' to 'c:\ccm_wa\hsaw~1'...
 1 project work area path(s) were updated:
     'hsaw~1': 'c:\ccm_wa\hsaw~1'
```

- Remove a directory with a leading special character from a path, using a leading backslash.

```
ccm wa /find "\\\+bill45" /replace ""  /reg /p hsaw~1
 Checking work area paths for replacement...
1 project(s) will be checked. Setting path for work area of 'hsaw~1' to
 'c:\ccm_wa\hsaw~1'...
 1 project work area path(s) were updated:
     'hsaw~1': 'c:\ccm_wa\hsaw~1'
```

In general, unless there are special characters or spaces in *find_str* or *new_str*, no quotes are needed (in a non-UNIX environment). The following examples show regular expressions without quotes.

- Convert a work area path from `c:\ccm_wa\bill\junk~1` to `c:\temp\ccm\bill45\junk~1`.

  ```
  ccm wa /find users\\bill\\ccm_wa\\ /replace temp\\ccm\\ /reg /p junk~1
   Checking work area paths for replacement...
   1 project(s) will be checked.
   Setting path for work area of 'junk~1' to
   'c:\temp\ccm\bill45\junk~1'...
   1 project work area path(s) were updated:
       'junk~1': 'c:\temp\ccm\bill45\junk~1'
  ```

  In this case, the only characters that must be escaped are the backslashes with which the regular expression processor normally initiates replacement constructs.

  One common reason not to escape the replacement construct is that UNIX file systems are case-sensitive.

- On UNIX, after executing a `ccm query` command that returned three projects with work areas, convert parts of the path names from lower case to first-character upper case.

  The need to do this might arise if a user working in the *ccm_admin* role changed the criteria for naming directories.

  ```
   pc-1: /users/bill/ccm_wa/owner/pc-1
   pi-1: /users/bill/ccm_wa/static/pi-1
   pw-1: /users/bill/ccm_wa/owner/pw-
  ```

  The arguments are enclosed in quotes to prevent the UNIX shell from processing `"*"` and other special characters, and the backslashes are escaped as well (although that is optional in this case). Parentheses identify the first and second expressions for later substitution. The selection set (`@`) operator refers to the results of the query.

  ```
  ccm wa -find "/users/bill/ccm_wa/([^/]+)/(.*)" -replace \
"/users/bill/ccm_wa/\\u\\1/\\2" -reg @
   Checking work area paths for replacement...
   3 project(s) will be checked.
   Setting path for work area of 'pc-1' to '/users/bill/ccm_wa/Owner/pc-
1'
   . . .
   Setting path for work area of 'pi-1' to '/users/bill/ccm_wa/Static/pi-
1'
   . . .
   Setting path for work area of 'pw-1' to '/users/bill/ccm_wa/Owner/pw-
1'
   . . .
   3 project work area path(s) were updated:
       'pc-1': '/users/bill/ccm_wa/Owner/pc-1'
       'pi-1': '/users/bill/ccm_wa/Static/pi-1'
       'pw-1': '/users/bill/ccm_wa/Owner/pw-1'
  ```

- On Windows, after executing a `ccm query` command that returned three projects with work areas, convert parts of the path names from first-character upper case to lower case.

  The need to do this might arise if the user working in the *ccm_admin* role changed the criteria for naming directories.

  ```
  pc-1: c:\bill\ccm_wa\Owner\pc-1
  pi-1: c:\bill\ccm_wa\Static/pi-1
  pw-1: c:\bill\ccm_wa\Owner/pw-1
  ```

  The arguments are enclosed in quotes to prevent the shell from processing `'*'` and other special characters, and the backslashes are escaped as well (although that is optional in this case). Parentheses identify the first and second expressions for later substitution. The selection set (`@`) operator refers to the results of the query.

  ```
  ccm wa /find "\\bill\ccm_wa\\([^\\]+)\\(.*)" /replace
"\\bill\ccm_wa\\\1\\\1\\\2" /reg @
   Checking work area paths for replacement...
   3 project(s) will be checked.
   Setting path for work area of 'pc-1' to 'C:\bill\ccm_wa\owner\pc-1'
   . . .
   Setting path for work area of 'pi-1' to 'C:\bill\ccm_wa\static\pi-1'
   . . .
   Setting path for work area of 'pw-1' to 'C:\bill\ccm_wa\owner\pw-1'
   . . .
   3 project work area path(s) were updated:
       'pc-1': 'C:\bill\ccm_wa\owner\pc-1'
       'pi-1': 'C:\bill\ccm_wa\static\pi-1'
       'pw-1': 'C:\bill\ccm_wa\owner\pw-1'
  ```

Work area conflicts

The reconcile process consists of two phases. The first phase, the reconcile operation, compares work area and database files, and if reconcile is able to resolve differences automatically, it does. No action is taken on files determined to be in conflict. The second phase, the conflict resolution phase, supplies you with a list of conflicts, and you determine how the conflicts are to be resolved. You can do this in batch mode (by resolving all conflicts in the same way), or on an individual basis (by selecting a unique resolution for the selected file). You can also leave the conflict unresolved.

Because the reconcile operation may cause files to be discarded, overwritten, or ignored, it is important to understand what happens when you perform a reconcile. It is also important to understand how conflicts are resolved.

The following information is presented in order for you to understand how a conflict is detected, and what happens when you select a specific method to resolve the conflict. The following topics are described:

- Conflict types
- How conflicts are resolved using batch mode
- Conflict resolution - update database from work area batch mode
- Conflict resolution - update work area from database batch mode
- Conflict resolution - manually selecting and resolving conflicts

Conflict types

The following information lists the eight types of work area conflicts and describes the situations in which they occur.

1. Work area change to working object.

   You have an object in the working state and make a work area change to that object, such as when you change a file in the file system.

2. Database change to working object.

   You have an object in the working state and the database gets updated by:
   - a location other than the work area where the working object is located
   - a different work area, because you may have two work areas
   - the database source cache file is updated outside of Rational Synergy's control

3. Work area change to static object.

   You change the permission of a file in the work area to a writable mode and then modify the file outside of Rational Synergy's control.

4. Database change to static object.

   The database source cache file for an object in a non-modifiable state is updated outside of Rational Synergy's control.

5. Object changed in multiple locations.

   Both the database source cache file for an object and the work area file for an object have been modified. The object could be in either a working or static state.

6. Files missing from the work area.

   The work area file for an object under Rational Synergy's control is missing from the work area. The object could be in either a working or static state.

7. Uncontrolled files.

   Files/directories in the work area are not under Rational Synergy's control.

8. Files in error.

   Files under Rational Synergy's control are in error; this type of error is usually related to link-based work areas. For this type of conflict, you do not have the ability to choose how these errors are resolved. They are handled specially by the reconcile process. The action taken can be:
   - update the work area from the database
   - update the database from the work area
   - relink the file (UNIX only)
   - delete the file.

   You can also choose not to resolve the conflicts.

How conflicts are resolved using batch mode

After conflicts have been detected, you can resolve them either on a case-by-case basis, or in batch mode.Two batch mode options are available: update database from work area, or update work area from database. When you use batch mode to resolve conflicts, all conflicts are resolved in the same manner. For example, if you select the option to update work area from database, all work area files are updated with files from the database.

The possible batch mode results are summarized in the following table.

Conflict Resolution Results - Batch Mode

Conflict Type	Result - Update database from work area batch mode	Result - Update work area from database batch mode
1. Work area change to working object	Database object updated with changes from work area	Work area file updated with file from database
2. Database change to working object	Database object updated with changes from work area	Work area file updated with file from database
3. Work area change to static object	Database object checked out, then updated with changes from work area	Work area file updated with file from database
4. Database change to static object	Database object checked out, then updated with changes from work area	Work area file updated with file from database
5. Object changed in multiple locations	Database object checked out, then updated with changes from work area	Work area file updated with file from database
6. Files missing from work area	Database objects not found in work area are "unused" in project	Missing files copied from the database to the work area
7. Uncontrolled files	Work area files added to database, then checked out	Uncontrolled files deleted from work area
8. Files in error	Action dependent on conflict type—you can specify only to resolve error	Action dependent on conflict type—you can specify only to resolve error

For more details, see Conflict resolution - update database from work area batch mode and Conflict resolution - update work area from database batch mode.

Conflict resolution - update database from work area batch mode

The batch mode `Update database from work area` resolves all conflicts by updating the database from the state of the work area. There is a conflict resolution strategy for each different conflict (see Conflict types).

The following list describes what happens when you resolve conflicts by selecting update database from work area batch mode.

1. Work area change to working object.

   The database object's source attribute is updated (work area changes accepted).

2. Database change to working object.

   The database object's source attribute is updated with the contents of the work area file (database changes discarded).

3. Work area change to static object.

   The database object is checked out and the source attribute is updated (work area changes accepted).

4. Database change to static object.

   The database object is checked out and the source attribute is updated with the contents of the work area file (discard database changes).

5. Object changed in multiple locations.

   The database object is checked out and the source attribute is updated with the contents of the work area file.

6. Files missing from the work area.

   An Unuse operation is performed on the database object associated with the missing work area file/directory.

7. Uncontrolled files.

   The files/directories are created in the project or directory where they are located. These files are in the *working* state.

8. Files in error.

   The action taken can be: update the work area from the database, update the database from the work area, relink the file (UNIX only), or delete the file.

   > **Note** Check-out operations can fail if it is not possible to check out the object; if this occurs, the work area file are left unchanged.

Conflict resolution - update work area from database batch mode

The batch mode `Update work area from database` resolves all conflicts by updating the work area from the state of the database. There is a conflict resolution strategy for each different conflict (see Conflict types).

The following list describes what happens when you resolve conflicts by selecting update work area from database batch mode.

1. Work area change to working object.

   The work area file is overwritten with the contents of the database object's source attribute (work area changes discarded).

2. Database change to working object.

   The work area file is overwritten with the contents of the database object's source attribute (database changes accepted).

3. Work area change to static object.

   The work area file is overwritten with the contents of the database object's source attribute (work area changes discarded).

4. Database change to static object.

   The work area file is overwritten with the contents of the database object's source attribute (database changes accepted).

5. Object changed in multiple locations.

   The work area file is overwritten with the contents of the database object's source attribute.

6. Files missing from the work area.

   The missing work area file/directory is copied from the database to the work area.

7. Uncontrolled files.

   The uncontrolled files/directories are deleted from the work area.

8. Files in error.

   The action taken can be: update the work area from the database, update the database from the work area, relink the file (UNIX only), or delete the file.

Conflict resolution - manually selecting and resolving conflicts

The reconcile operation identifies the files it finds in conflict, but takes no action to resolve the conflict unless instructed to do so by you.

The manual selection mode "`Select`" allows you to choose how the conflicts are resolved. This mode allows you to use different resolutions to resolve conflicts. Depending on the type of conflict detected, you can update the database from the work area, update the work area from the database, merge the files in conflict, or ignore the conflict. For more information about resolving conflicts using this method, use help for the specific type of conflict resolution.

Links to all Rational Synergy help

The following links enable you to read any of the Rational Synergy Help systems in PDF:

- Rational Synergy Help, Developers PDF
- Rational Synergy Help, Team Leads and Build Managers PDF
- Rational Synergy CLI Help, Web mode| PDF
- Synergy Classic GUI Help, (UNIX, Windows) PDF

Notices

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

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

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send written inquiries to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions. Therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

Intellectual Property Dept. for Rational Software
IBM Corporation
1 Rogers Street
Cambridge, Massachusetts 02142
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

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

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Trademarks

See http://www.ibm.com/legal/copytrade.html.

Microsoft, Windows, and/or other Microsoft products referenced herein are either trademarks or registered trademarks of Microsoft Corporation.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product or service names may be trademarks or service marks of others.

Index

Symbols

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W