Rational. Statemate

IBM

Configuration Management

Rational Statemate Configuration Management

This edition applies to IBM® Rational® Statemate® 4.6 and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

Configuration Management Tool

Rational® Statemate® ncludes a Configuration Management (CM) tool with built-in “checkin-checkout” capabilities. However, you might transparently substitute the revision engine and repository format of a third-party configuration management tool. Interface modules are available for the following widely-used, third-party configuration management tools:

- Continuus Software Corp. Continuus/CM
- Intersolv PVCS Version Manager
- Rational® Software® ClearCase®

Note

To use ClearCase as the CM tool, install Rational Clear Case and set it as the Source Code Control (SCC) provider.

Set STM_SCC_CLEARCASE environment variable, and when creating a project, select SCC in the CM Tool field.

On Windows, third-party tools are supported by Source Code Control (SCC), a common interface introduced by Microsoft.

If you want Rational Statemate to work with a different CM tool, an add-on module is available that allows you to create a script-based interface to your CM tool of choice.

Note

- When using the Rational Statemate interface with a third-party CM tool, you cannot use the Rational Statemate interface to modify or delete Rational Statemate project files. You cannot use the third-party CM tool to check in or check out files from a repository used to hold a Rational Statemate project. You might use the Rational Statemate interface for read-only operations such as viewing, searching, and reporting.
- Rational Statemate does not support checking out different revisions of the same file. Only one revision can be checked out at a time.

Rational Statemate CM Interface

Regardless of which CM engine and repository you use, the configuration management operations in the Rational Statemate user interface remain almost exactly the same. The intent is to make the underlying mechanism as transparent to your project members as possible.

Note

Only few operations are supported by Rational Statemate, which is not intended to serve as a full user interface for any third-party CM tool. Many operations available in third-party tools, such as comparing or merging versions, cannot be performed within Rational Statemate.

The remainder of this section provides a brief overview of the user interface to the Rational Rational Statemate built-in configuration management tool. Subsequent sections describe how to replace the built-in facility with third-party tools.

The Project Databank

Each Rational Statemate project has a common repository area called the *databank*, which contains the charts and files belonging to the entire project. When a new project is created, the databank can be placed in any directory to which the project manager has read and write access. Many sites designate a special location for project databanks; check with your system administrator.

Configuration Items

Elements stored in the databank are called *configuration items* and are stored as ASCII files. For example, configuration items are:

- Charts - Statecharts, Activity charts, Flowcharts, Use Case Diagrams, Sequence Diagrams, Module Charts, Continuous Diagrams, and Global Definition Sets (GDSs)
- Analysis profiles
- Simulation Control Language (SCL) files
- Waveform profiles
- Status files
- Check Model profiles
- Code generation profiles
- Documentor (DGL) templates
- Include files
- Configuration files
- Panels
- Components
- Targets and Cards

Locking

Rational Statemate uses the standard “lock” paradigm to prevent configuration items from being modified by more than one person at any time. When checking an item out, the following occurs:

- The project member can optionally lock the item, preventing anyone else from modifying it. Other project members continue to have read access to the item but they cannot lock it.
- A copy of the item is created in the project member’s designated disk space, which is called a *workarea*. Any project member can modify a copy of an item in their workarea, but only the project member who holds the lock on the item can check it in.

When checking an item in, the following occurs:

- The project member can choose to retain or release the lock. Until the lock is released, no one else can modify the item. When more than one project member needs to modify an item, it is important to unlock the item.
- The project member can choose to save or delete the copy of the item in the workarea.

Version Numbers

Each configuration item has a version number. Typically, the highest version number represents the current working design. Lower version numbers represent earlier designs. Each time a project member checks an item in, Rational Statemate assigns it a version number by incrementing the highest existing version number.

The Rational Statemate built-in CM tool tracks versions using whole numbers (positive integers). When you create a new item and check it into the databank, Rational Statemate assigns it a “1” version number. If a third-party CM tool uses a different system of version numbering, the version numbers displayed by Rational Statemate conform to the format of the third-party tool.

Protection Levels and Groups

The databank directory structure is created with “world read/write,” permissions but specific items within the structure can be protected differently. You can assign one of the following protection levels (common to most operating systems) to configuration items in the databank:

- **None -** The item cannot be checked out.
- **Read-Only -** The item can be checked out without a lock and used or modified, but cannot be checked in.
- **Read-Write -** The item can be checked out with a lock (guarantees that other users cannot change it while you are working on it) and checked in.

You can assign a protection level to the following groups:

- **Owner -** The project member who first created the configuration item in the databank.
- **Group -** Project members who belong to the same group as the owner (according to the definitions set by the operating system).
- **Others -** All project members.

All versions of a configuration item belong to the same owner and have the same protection level.

The Databank

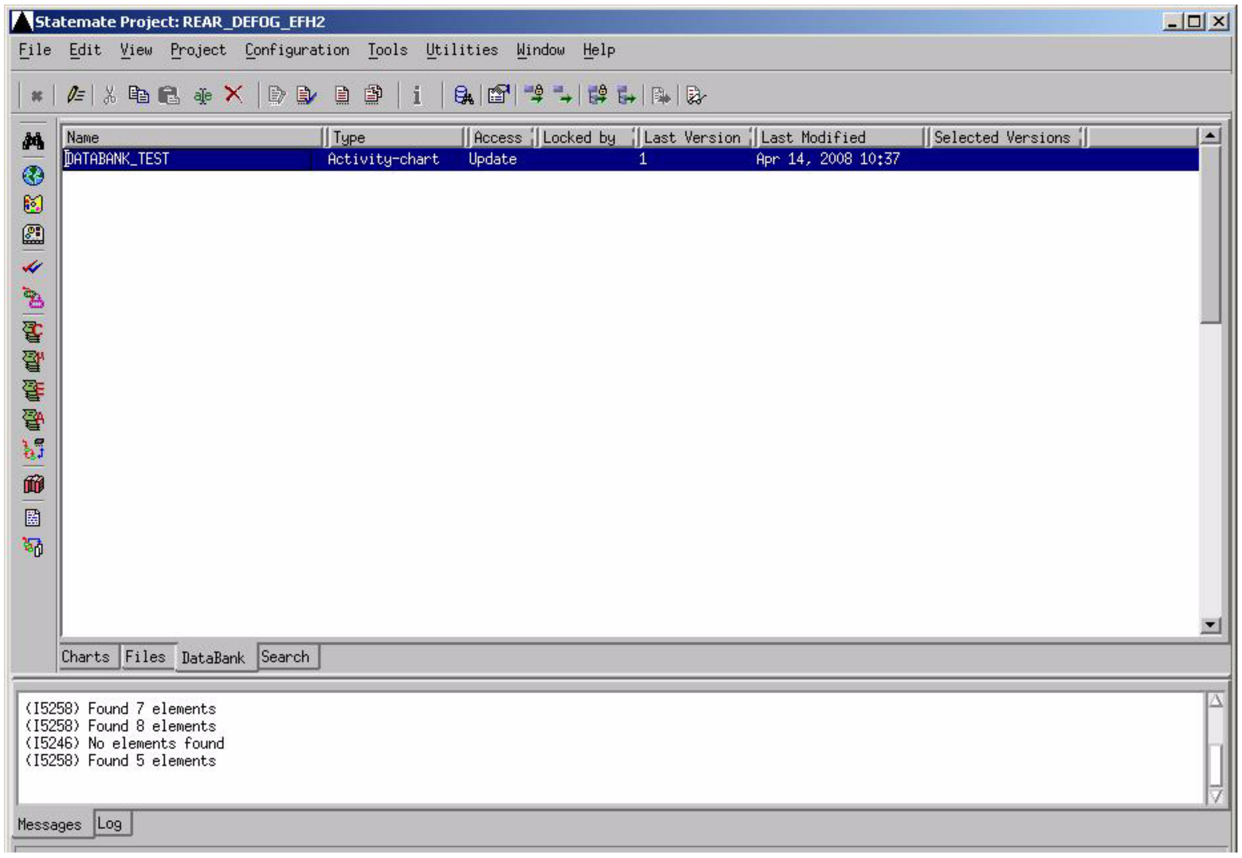
The Databank allows you to graphically navigate through the information in your databank.

Starting and Using the Databank

In the main project window, click the DataBank tab. The Databank main window opens. Select **View > Filter** to choose the types of charts and files you want to be listed in the Databank main window and click **OK**.

Databank Main Window

The Databank main window displays a list of the configuration items in the databank, along with information about the selected item.



Name	Type	Access	Locked by	Last Version	Last Modified	Selected Versions
DATABANK_TEST	Activity-chart	Update		1	Apr 14, 2008 10:37	

```
(I5258) Found 7 elements
(I5258) Found 8 elements
(I5246) No elements found
(I5258) Found 5 elements
```

Configuration Item List

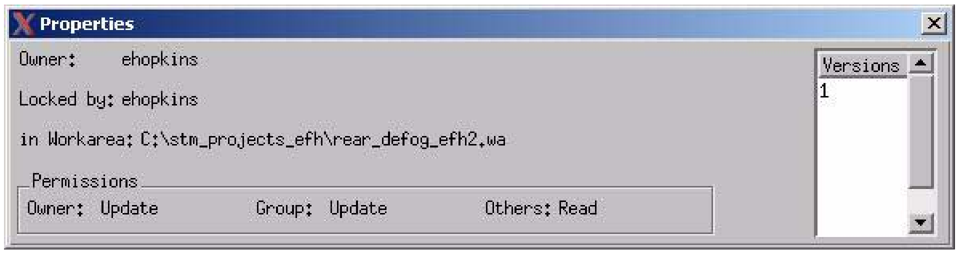
The Configuration Item List displays a list of the configuration items in the databank, as specified in the Show Files dialog:

- The **Name** of the configuration item.
- The **Type** of item (Statechart, Activity-chart, Configuration file, and so forth).
- The **Access** (protection level) consisting of one of the following values: none, read, update.
- **Locked by** (the name of the person who checked it out).
- The number of the **Last Version** to be checked in (a designation of `.r` indicates that the version is part of a release).
- **Last Modified** is the date the file was checked into the DataBank.
- The **Selected Version** on which various operations can be performed.

Item Properties

Double-click on a single configuration item and the **Selected Item Information** shows more detailed information, including a list of all existing versions.

- The name of the **Owner** of the item.
- Whom, if anyone, the item is **Locked by** (the name of the person who checked it out).
- The name of the **Workarea** containing the item.
- The **Permissions** (access) available to the Owner, the project group, and all others.


Properties

Owner: ehopkins

Locked by: ehopkins

in Workarea: C:\stm_projects_efh\rear_defog_efh2.wa

Permissions

Owner: Update Group: Update Others: Read

Versions

1

Existing Version List

In the Existing Version List, you can select a specific version (or versions) on which to apply Configuration menu operations.

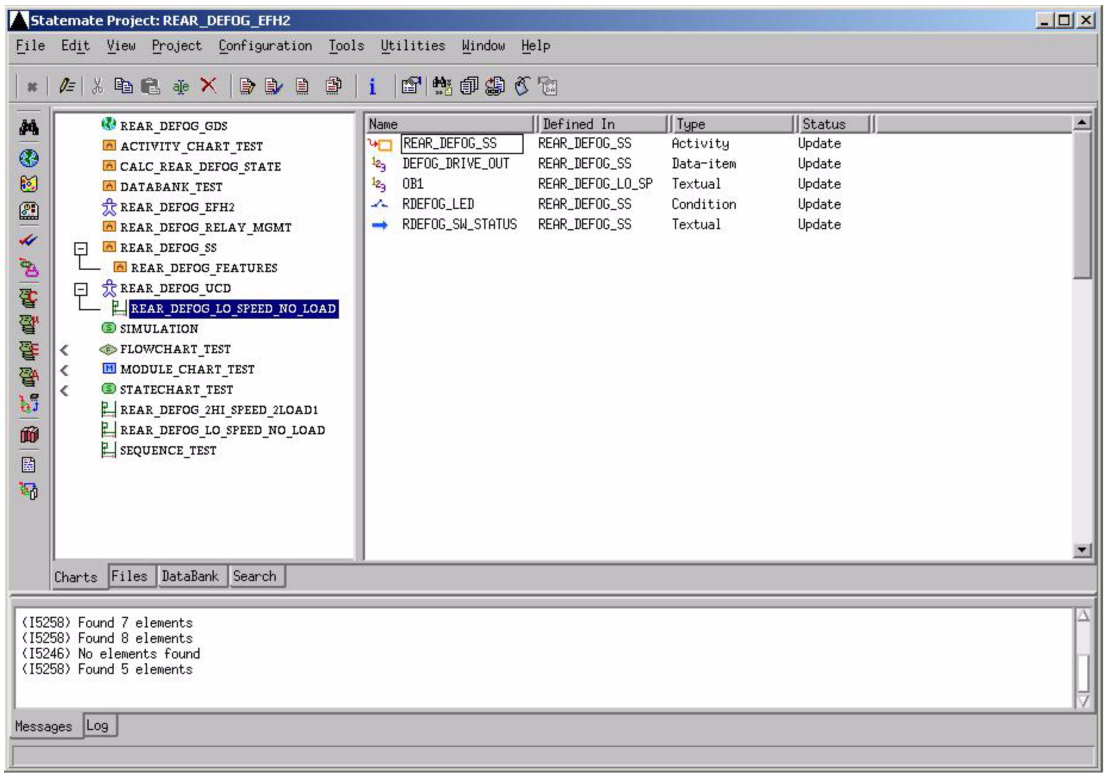
The Workarea

The Workarea allows you to graphically navigate through the information in your workarea. In the main project window, select the Charts tab. The Workarea main window appears.

In the upper area, the main window displays all the charts in your workarea arranged in a hierarchical tree, making it easy for you to see their relationships.

Select the Files tab and the main window displays a list of the charts and other configurable items in your workarea, along with information about them, such as type and the checked-out version's mode, number, and modification status.

In **update** mode, all editing, viewing, and tool launch features are active (shown in the following figure). In **read-only** mode, all viewing and tool launch features are active. The editing features and drawing icons are disabled.



Name	Defined In	Type	Status
REAR_DEFOG_SS	REAR_DEFOG_SS	Activity	Update
DEFOG_DRIVE_OUT	REAR_DEFOG_SS	Data-item	Update
OB1	REAR_DEFOG_LO_SP	Textual	Update
RDEFOG_LED	REAR_DEFOG_SS	Condition	Update
RDEFOG_SW_STATUS	REAR_DEFOG_SS	Textual	Update

```
(I5258) Found 7 elements
(I5258) Found 8 elements
(I5246) No elements found
(I5258) Found 5 elements
```

Configuration Management Operations

Most of the Rational Statemate Configuration Management operations are common to both the Databank and the Workarea. For example, you can check items out from either the Databank or the workarea. Some operations, however, are unique to one or the other. For example, you only can check items in using the Workarea.

The following table show the configuration operations that can be performed in the databank and the workarea.

Menu Operation	Databank	Workarea	Description
Check In to Databank	No	Yes	Saves the current version of the selected configurable items in the databank and updates their current revision numbers. Options are: • Hold & Keep Lock • Hold & Release Lock • Check in & Delete
Check In to Databank with Descendants	No	Yes	Saves the current version of the selected configurable items and descendants in the databank and updates their current revision numbers. Options are: • Hold & Keep Lock • Hold & Release Lock • Check in & Delete
Check Out **Check Out Chart/File**	Yes	Yes	Copies and locks (if ‘with lock’ is selected) files/charts from the databank to your workarea (update mode). Options are: • With Lock • Without Lock
Check Out With Descendants	Yes	Yes	Copies and locks (if ‘with lock’ is selected) files/charts and descendants from the databank to your workarea (update mode). Options are: • With Lock • Without Lock
Update Workarea	No	Yes	Opens Update Workarea dialog box.

Menu Operation	Databank	Workarea	Description
Advanced Update Workarea	No	Yes	Options are: • Remove Unused Elements • Filtered Check Out from Databank of Selected • Filtered Check Out from Databank of All • Filtered Check Out of GDSs of Selected Charts • Filtered Check Out Parent from Databank
Check Out Parent from Databank	Yes	Yes	Copies parent charts from the databank to your workarea
Release Lock	Yes	Yes	Unlocks files/charts in the databank that currently exist in your workarea (set read-only mode).
Release Lock Notification	Yes	No	Notifies you when a lock, set by another user on a chart/file, is released
Lock	Yes	No	Locks files/charts in the databank that currently exist in your workarea (sets update mode).
Purge	Yes	No	Deletes older versions of files/charts in the databank that you own.
Delete	Yes	No	Deletes files/charts in the databank that you own.
Create Configuration	No	Yes	Opens the Create Configuration dialog box.
Execute Configuration	Yes	Yes	Checks-out all charts/files in the configuration file into the workarea.
Create/Modify Component	No	Yes	Allows you to set up a component from a selected generic chart or an existing component.
Remove Component	No	Yes	Allows you to remove a component from the workarea.
Component Versions	Yes	Yes	Displays the versions of a component configuration file.

Using Configuration Management

A Configuration file is a “snapshot” of the current workarea, including all configuration item versions. Configuration files are used primarily for capturing milestones or releases of a project. Other users can execute a Configuration file to load the same items (and versions) that you have in your workarea.

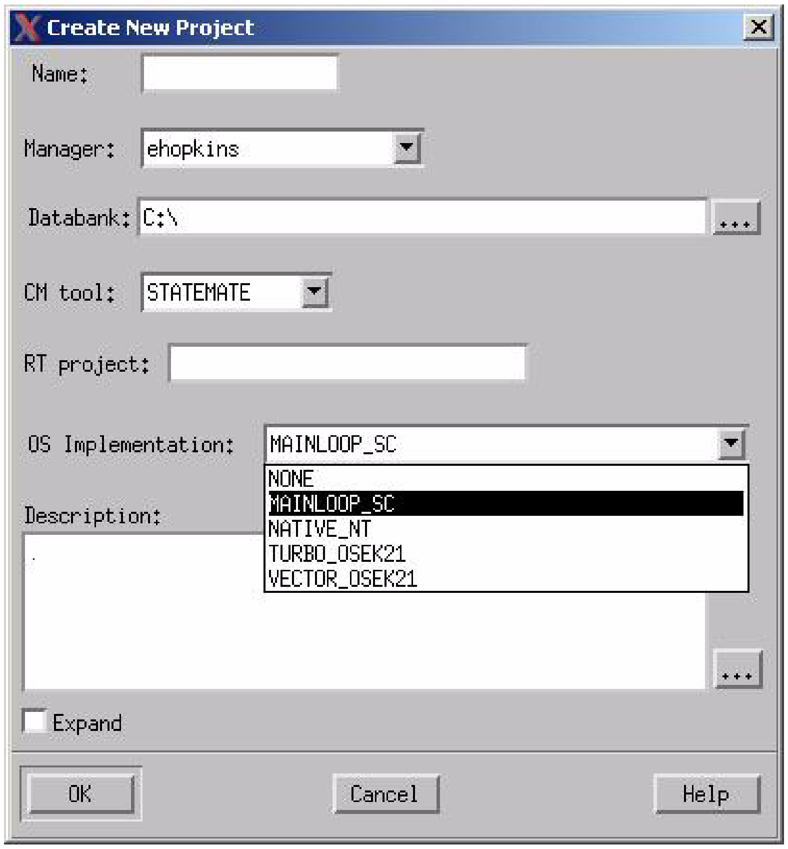
Configuration files are also useful when you upgrade to a new major Rational Statemate release.

Menu Operation	Databank	Workarea	Description
Create Configuration	No	Yes	Creates a Configuration file that captures the current workarea. You can include all files or only selected files. At the same time, you can check in the new Configuration file and/or the files that are in the configuration itself.
Execute Configuration	Yes	Yes	Selects a previously created Configuration file and loads charts/files into your workarea. The Configuration file must exist in the workarea. If it does not, check the Configuration file out of the databank.

Selecting a CM Tool for a Rational Statemate Project

The decision to use the Rational Statemate built-in facility or a third-party configuration management tool must be made when you create a Rational Statemate project. You cannot change the CM tool used by an existing project. You can, however, create a new project from an existing project by importing all the objects.

The **File > New Project** dialog box contains a control named **CM tool,** as shown in the following figure. Select an available tool from the **CM Tool** pull-down menu, as shown here.


Create New Project

Name:

Manager: ehopkins

Databank: C:\

CM tool: STATEMATE

RT project:

OS Implementation: MAINLOOP_SC

You can specify the CM tool used by default for new projects in the **Project > General Preferences** dialog box.

Defining Third-Party CM Tool Interfaces

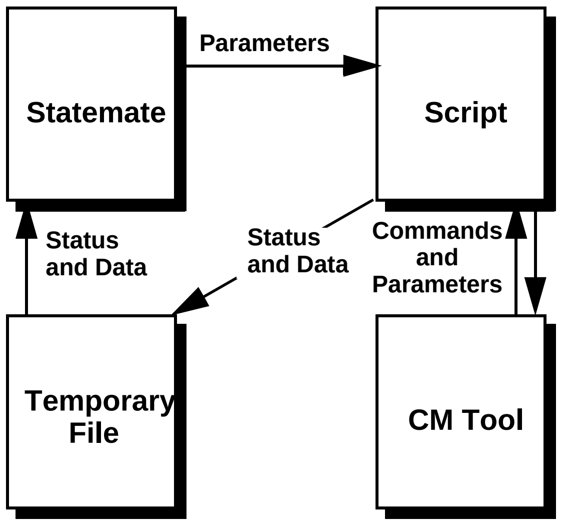
In addition to supported interface modules for certain widely-used CM tools, modules are available that allow you to define interfaces to other CM tools. This section explains how to define a new Rational Statemate interface to any third-party CM tool.

Note

When using the Rational Statemate interface to a third-party CM tool, you cannot use the tool interface to modify or delete Statemate project files. You cannot use the third-party CM tool to check in or check out files from a repository used to hold a Statemate project. You can use the tool interface for read-only operations such as viewing, searching, and reporting.

Script-Based Interface to CM Tools

When using a third-party tool, the CM operations available in the Rational Statemate Databank and Workarea are handled by external executable files. Statemate does not directly communicate with your CM tool. Instead, it provides a generic CM interface. You provide the files that map each operation in the Rational Statemate generic interface to some specific CM tool. For convenience, these files are referred to simply as *scripts* in this section. Statemate provides a sample implementation that you can copy. The flow of information between Statemate, the scripts, and the CM tool is as follows:



Since Rational Statemate simply issues requests to the operating system to execute files, these scripts can be implemented using whatever programming resources are available to you. For example, you can use the following:

- Scripts, written in an interpreted language such as Perl or C shell, that communicate with your CM tool through its command line interface.
- Programs, written in a compiled language such as C or C++, that communicate with your CM tool through its application programming interface.

You must provide a script for each of the CM operations described in this section. You can name them and place them anywhere. However, you must also provide a file that maps the Rational Statemate CM operations to your scripts, as explained in the sections that follow.

How Rational Statemate Determines Available CM Tools

In Rational Statemate dialog boxes, the list of CM tools available is composed of:

- STM, the built-in facility, which is always available.
- Any other CM tool for which there is a **mapping file.**

A mapping file is a text file named:

```
$STM_ROOT/etc/cmt/cm_tool_name.cmt
```

Replace `cm_tool_name` with your own CM tool's name. For example, if the directory `$STM_ROOT/etc/cmt/` contains the following mapping files:

```
cm_toolA.cmt
cm_toolB.cmt
cm_toolC.cmt
```

In Rational Statemate dialog boxes, the CM Tool control shows the following tools

Statemate
cm_toolA
cm_toolB
cm_toolC

To add another CM tool to the list, simply create a new `cm_tool_name.cmt` file by copying the template file `cm.cmt_template`.

Mapping Individual CM Operations to Scripts

The mapping file `cm_tool_name.cmt` is a text file that maps each CM operation in Rational Statemate to a script that actually performs the operation. The contents of the mapping file template are shown in the following table:

CM_USER_FUNC_init	path_of_scripts/init
CM_USER_FUNC_get_file	path_of_scripts/get_file
CM_USER_FUNC_put_file	path_of_scripts/put_file
CM_USER_FUNC_is_file_in_bank	path_of_scripts/is_file_in_bank
CM_USER_FUNC_is_locked_in_bank	path_of_scripts/is_locked_in_bank
CM_USER_FUNC_get_ver_from_bank	path_of_scripts/get_ver_from_bank
CM_USER_FUNC_unlock_file	path_of_scripts/unlock_file
CM_USER_FUNC_lock_file	path_of_scripts/lock_file
CM_USER_FUNC_delete_from_bank	path_of_scripts/delete_from_bank
CM_USER_FUNC_delete_revision	path_of_scripts/delete_revision
CM_USER_FUNC_calc_archive_name	path_of_scripts/calc_archive_name
CM_USER_FUNC_get_pattern	path_of_scripts/get_pattern
CM_USER_FUNC_get_locked_by	path_of_scripts/get_locked_by
CM_USER_FUNC_get_versions	path_of_scripts/get_versions
CM_USER_FUNC_modify_to_archive	path_of_scripts/modify_to_archive

Note

The list of operations in this file is defined by Rational Statemate. You cannot add or remove operations.

To define a new interface for a CM tool, edit the new mapping file and replace the string `path_of_scripts` with the actual path of the directory that contains the scripts for your CM tool. It is recommended that you do not change the names of the scripts. Creating the scripts is described in the next section.

Creating CM Operation Scripts

As stated previously, the scripts that implement individual CM operations can be:

- Shell scripts or Perl scripts that communicate with the CM tool via its command line interface.
- Executable programs that communicate with the CM tool via its application programming interface.
- Any other implementation method available to you.

When Rational Statemate issues a request to the operating system to execute a specific script, it passes parameters containing the information needed to perform the task. The script reads the parameters from the argument vector. For example, this Perl statement stores the parameters in a list variable:

```
#!/usr/bin/perl -w
($working_dir, $bank_dir, $tmp_result_file) = @ARGV;
```

It is recommended that you write the parameters to the standard output to assist in debugging. For example:

```
$func = "init";
$echo = "$func : $working_dir,
$bank_dir, $tmp_result_file\n";
print $echo;
```

Then, the script calls the CM tool to perform the operation.

When the CM tool finishes, the script returns the result of the CM tool operation (and in some cases data) to Rational Statemate by creating a temporary file. Rational Statemate deletes the temporary file after reading it.

Parameter Types

The following table defines the set of parameter types used by Rational Statemate to communicate with scripts

Parameter	Description
bank_dir	Full pathname of the databank directory.
ext	Extension of the configuration item in the Workarea.
full_archive_name	Full pathname of the archive file of the configuration item in the databank.
lock	lock state: 1 for “with lock”, 0 for “without lock”.
name	Name of the configuration item in workarea (without extension).
ret_val	Status: 1 for success, 0 for failure.
revision	Version number in the CM tool’s own format.
temp_result_file	The name of a temporary file in which Rational Statemate expects the script to return its results; Rational Statemate deletes this file after reading it.
tmp_file_to_ci	Temporary file of chart to be checked in.
workarea_dir	Full pathname of the workarea.
workfile_name	Name of the configuration item in the workarea (including extension).
working_dir	Full pathname of the of the working directory within the workarea.

The following sections describe the specific scripts and input/output parameters.

CM Script Reference

This section provides examples in the form of Perl scripts for an interface to RCS, the UNIX Revision Control System.

calc_archive_name

Composes the full archive name of a configuration item.

Syntax:

```
CM_USER_FUNC_calc_archive_name
```

input	bank_dir
	name
	ext
	temp_result_file
output	ret_val
	full_archive_name

delete_from_bank

Deletes an archive file from the databank.

Syntax:

```
CM_USER_FUNC_delete_from_bank
```

input	bank_dir
	name
	ext
	full_archive_name
	temp_result_file
output	ret_val

delete_revision

Deletes a specified revision of an archive file in the databank.

Syntax:

```
CM_USER_FUNC_delete_revision
```

input	bank_dir name ext full_archive_name revision temp_result_file
output	ret_val

get_file

Checks out a file from the archive.

Syntax:

```
CM_USER_FUNC_get_file
```

input	working_dir workfile_name full_archive_name revision lock temp_result_file
output	ret_val revision

Example:

```
#!/usr/bin/perl -w


#initialize variables

($working_dir , $workfile_name , $full_archive_name , $revision ,
$lock ,$tmp_result_file) = @ARGV;

#echo command

$func = "get_file";

$echo = "$func : $working_dir , $workfile_name , $full_archive_name ,
$revision , $lock , $tmp_result_file\n";

print $echo;

#update result file

open(RESULT_FILE , "> $tmp_result_file");


if ($lock == 0)

  {

  `cd $working_dir ; co $full_archive_name`;

  }

else

  {

  `cd $working_dir ; co -l $full_archive_name`;

  }

print RESULT_FILE "1\n";

$str = `cd $working_dir ; rlog $full_archive_name | grep head`;

$out_ver = substr($str,10,100);

print RESULT_FILE "$out_ver\n";
```

get_locked_by

Returns the name of the user who locked an archive file.

Syntax:

```
CM_USER_FUNC_get_locked_by
```

input	full_archive_name temp_result_file
output	ret_val locked_by

get_pattern

Returns a pattern that can be used to search the databank for archive files of a specific type.

Syntax:

```
CM_USER_FUNC_get_pattern
```

input	ext temp_result_file
output	ret_val pattern

Example:

Type of File	Pattern
Statechart Description	`^.*\\.sch\\.desc$`
ClearCase Statechart Archive	`^.*\\.sch$`
RCS Statechart Archive	`^.*\\.sch,v\$`

get_ver_from_bank

Gets the top revision of the file in the databank.

Syntax:

```
CM_USER_FUNC_get_ver_from_bank
```

input	bank_dir name ext full_archive_name temp_result_file
output	ret_val revision

get_versions

Returns a list of versions of an archive file (in CM tool format).

Syntax:

```
CM_USER_FUNC_get_versions
```

input	full_archive_name temp_result_file
output	ret_val [oldest_revision ...] newest_revision

init

Initiates the CM tool.

Syntax:

```
CM_USER_FUNC_init
```

input	workarea_dir bank_dir temp_result_file
output	ret_val

Example:

```
#!/usr/bin/perl -w
#initialize variables
($working_dir , $bank_dir , $tmp_result_file) = @ARGV;
#echo command
$func = "init";
$echo = "$func : $working_dir , $bank_dir , $tmp_result_file\n";
print $echo;
#perform operation
#update result file
open(RESULT_FILE , "> $tmp_result_file");
print RESULT_FILE "1\n";
```

is_file_in_bank

Checks whether or not a file exists in the databank (has been checked in before).

Syntax:

```
CM_USER_FUNC_is_file_in_bank
```

input	full_archive_name temp_result_file
output	ret_val

is_locked_in_bank

Checks whether or not a file in the databank is locked (was checked out with lock).

Syntax:

```
CM_USER_FUNC_is_locked_in_bank
```

input	full_archive_name temp_result_file
output	ret_val

lock_file

Executes the lock command on the file in the databank.

Syntax:

```
CM_USER_FUNC_lock_file
```

input	bank_dir name ext full_archive_name temp_result_file
output	ret_val

modify_to_archive

Converts a file name into an archive full path name (similar to calc_archive_name).

Syntax:

```
CM_USER_FUNC_modify_to_archive
```

input	file_name temp_result_file
output	ret_val archive_file_name

put_file

Checks in a file to the archive.

Syntax:

```
CM_USER_FUNC_put_file
```

input	working_dir workfile_name tmp_file_to_ci ext temp_result_file
output	ret_val revision

unlock_file

Executes the unlock command on a file in the databank.

Syntax:

```
CM_USER_FUNC_unlock_file
```

input	bank_dir name ext full_archive_name temp_result_file
output	ret_val

User Function Interface to CM Tool

There is an API in the Rational Statemate CM DLL called `STM_CM_init_item`. This API is called by the tool on any CM item, before any other call is done on this item, and only once per configuration item per session.

This API is optional. The tool calls it only if it is implemented in the specific DLL used.

Define the Dynamic-Library path name in the `cm_tool_name`.cmt file located in the directory `$$STM-ROOT/etc/cmt/`. For example:

```
DLL-NAME /root31/bin/stm_cm_user.dll
```

Note

The DLL path name should be all lowercase.

```
API functions (see stm_cn_user,h)
#define STM_CM_success             0
#define STM_CM_message_length      1024
#define STM_CM_archive_name_length 2048
#define STM_CM_version_length      80
#define STM_CM_user_name_length    512
```

When the return value of a certain API is of type “int”, the following convention holds:

- “0” (zero) marks success
- Anything but “0” is regarded as error-code

Memory management is done independently between STMM and the dynamic-library:

- When STMM calls a certain API with parameters defined as “const char *”, the intention is that those are managed by STMM and should be regarded as read-only variables.
- When STMM calls a certain API with parameters defined as “char [length]” (with length being either 512 or 1024), the intention is for the API to use the parameters as error messages and returned information.
- When STMM calls a certain API with parameters defined as “char * []” (as in 2.n, 2.p), the intention is that those are managed by the dynamic-library. STMM copies the content as soon as the API returns. It is recommended to use static buffers in the dynamic-library implementation for those parameters.

For building the DLL, you can use stm_cm_user.def file:

```
LIBRARY
stm_cm_user
EXPORTS
```

The following sections describe the user functions:

- STM_CM_init
- STM_CM_calc_archive_name
- STM_CM_get_files_list
- STM_CM_put_file
- STM_CM_is_locked_in_bank
- STM_CM_is_file_in_bank
- STM_CM_unlock_file
- STM_CM_lock_file
- STM_CM_standalone_lock_file
- STM_CM_get_ver_from_bank
- STM_CM_get_versions
- STM_CM_get_locked_by
- STM_CM_delete_from_bank
- STM_CM_delete_revision
- STM_CM_modify_to_archive
- STM_CM_get_pattern
- STM_CM_get_files_list
- STM_CM_rollback
- STM_CM_get_file_last_modified_date
- STM_CM_init_item
- STM_CM_close
- STM_CM_begin_databank_operation
- STM_CM_end_databank_operation

STM_CM_begin_databank_operation

Starts the databank operation.

Syntax:

```
int
STM_CM_begin_databank_operations(
   char error_messages[STM_CM_message_length])
```

STM_CM_calc_archive_name

Composes the full archive name of a configuration item.

Syntax:

```
STM_CM_calc_archive_name
```

input	bank_dir
	name
	ext
output	

```
int
STM_CM_calc_archive_name(
   const char * bank_dir,
   const char *name,
   const char *ext,
   char full_archive_name[STM_CM_archive_name_length],
   char error_message[STM_CM_message_length]
);
```

STM_CM_close

Closes the databank.

Syntax:

```
int
   STM_CM_close(
   char error_message[STM_CM_message_length
);
```

STM_CM_delete_from_bank

Deletes an archive file from the databank.

Syntax:

```
STM_CM_delete_from_bank
```

Input	bank_dir name
Output	

```
int
STM_CM-delete_from_bank(
   const char *bank_dir,
   const char *name,
   const char *ext,
   const char *full_archive_name,
   char error_message[STM_CM_message_length]
);
```

STM_CM_delete_revision

Deletes a specified revision of an archive file in the databank.

Syntax:

```
STM_CM_delete_revision
```

Input	bank_dir name ext full_archive_name revision
Output	

```
int
STM_CM_delete_revision(
   const char *bank_dir,
   const char *name,
   const char *ext,
   const char *full_archive_name,
   const char *revision,
   char error_message[STM_CM_message_length]
);
```

STM_CM_end_databank_operation

Ends the databank operation.

Syntax:

```
int
   STM_CM_end_databank_operation(
   char error_message[STM_CM_message_length]
);
```

STM_CM_get_file

Checks out a file from the archive.

Syntax:

```
STM_CM_get_file
```

Input	working_dir workfile_name full_archive_name user_name revision
Output	

```
int
STM_CM_get_file(
   const char * working_dir,
   const char *workfile_name,
   const char *full_archive_name,
   const char *user_name’
   const char *revision,
   int lock,
   char out_revision[STM_CM_version_length],
   char error_message[STM_CM_message_length]
);
```

STM_CM_get_files_list

This function returns the `list_of_files` into the `bank_dir`. The size of the `list_of_files` is then written in the `number_of_files`.

If there are more than the `number_of_files` files in the `bank_dir`, a fail status is returned, and the `number_of_files` is changed to contain the actual number of files in the `bank_dir`. A second call to the API is than performed with the actual size for the `list_of_files`.

Syntax:

```
int
STM_CM_get_files_list(
   const char *bank_dir,
   const char *pattern,
   int *number_of_files,
   char *list_of_files[],
   char error_message[1024])
);
```

Note

Do NOT allow your code to write more than the given `number_of_files` into the `list_of_files` parameter, or it will cause memory access violations.

STM_CM_get_file_last_modified_date

Retrieve the last modification date of chart/file

Syntax:

```
int
STCM_CM_get_file_last_modified_date(
   const char *full_archive_name,
   char_last_modified_date_info[STM_CM_version_length],
   long *date_num_info,
   char error_meassage[STM_CM_message_length]
);
```

STM_CM_get_locked_by

Returns the name of the user who locked an archive file.

Syntax:

```
STM_CM_get_locked_by
```

Input	full_archive_name revision
Output	

```
int
STM_CM_gte_locked_by(
   const char *full_archive_name,
   const char *revision,
   char locked_by[STM_CM_user_name_length],
   char error_message[STM_CM_message_length]
);
```

STM_CM_get_pattern

Returns a pattern that can be used to search the databank for archive files of a specific type.

Syntax:

```
STM_CM_get_pattern
```

Input	ext
Output	

```
int
STM_CM_get_pattern(
   const char *ext,
   char pattern [STM_CM_message_length],
   char error_message[STM_CM_message_length]
);
```

STM_CM_get_ver_from_bank

Gets the top revision of the file in the databank.

Syntax:

```
STM_CM_get_ver_from_bank
```

Input	bank_dir name ext full_archive_name
Output	

```
int
STM_CM_get_ver_from_bank(
   const char * bank_dir,
   const char *name,
   const char *ext,
   const char *full_archive_name,
   char revision[STM_CM_version_length],
   char error_message[STM_CM_message_length]
);
```

STM_CM_get_versions

Returns a list of versions of an archive file (in CM tool format).

Syntax:

```
STM_CM_get_versions
```

Input	full_archive_name number_of_versions versions
Output	

```
int
STM_CM_get_version(
   const char *full_archive_name,
   const char *number_of_versions,
   const char *versions[],
char error_message[STM_CM_message_length]
);
```

Note

When the API is called, the `number_of_versions` parameter specifies the number of possible entries in the `versions` parameter. The API is expected to set this parameter to the actual number of revisions. If this number is more than the number of possible entries in the `versions` parameter, the API is called again with the appropriate number of possible entries. If more entries are needed, the function teturns a “fail” status. You cannot specify more than the given `number_of_versions` value in the `versions` parameter because it causes memory access violations.

STM_CM_init

Initiates the CM tool.

Syntax:

```
STM_CM_init
```

Input	workarea_dir bank_dir
Output	

```
int
STM_CM_init(
   const char *workarea_dir,
   const char * bank_dir,
   char error_message[STM_CM_message_length]
);
```

STM_CM_init_item

The CM DLL calls this API before any other call is sent to an Item. This occurs once per configuration item per session. This API is optional in that the tool calls it only if it is implemented in the specific DLL used.

Syntax:

```
int
STM_CM_init_item(
   const char *bank_dir,
   const char *name,
   const char *ext,
   char error_message[STM_CM_message_length]
);
```

STM_CM_is_file_in_bank

Checks whether or not a file exists in the databank (has been checked in before).

Syntax:

```
STM_CM_is_file_in_bank
```

Input	full_archive_name is_in_bank
Output	

```
int
STM_CM_is_file_in_bank(
    const char *full_archive_name,
    int * is_in_bank,
    char error_message[STM_CM_message_length]
);
```

STM_CM_is_locked_in_bank

Checks whether or not a file in the databank is locked (was checked out with lock).

Syntax:

```
STM_CM_is_locked_in_bank
```

Input	full_archive_name revision is_locked

```
int
STM_CM_is_locked_in_bank(
    const char *full_archive_name,
    const char *revision,
    int *is_locked
    char error_message[STM_CM_message_length]
);
```

STM_CM_lock_file

Executes the lock command on the file in the databank

Syntax:

```
STM_CM_lock_file
```

Input	bank_dir name ext full_archive user_name version

```
int
STM_CM_lock_file(
   const char *bank_dir,
   const char *name,
   const char *ext,
   const char *full_archive_name,
   const char *user_name,
   const char *version
   int in_load_op,
   char error_message[STMCM_message_length]
);
```

STM_CM_modify_to_archive

Converts a file name into an archive full path name (similar to **calc_archive_name**)

Syntax:

```
STM_CM_modify_to_archive
```

Input	file_name
Output	

```
int
STM-CM_modify_to_archive(
   const char *file_name,
   char out_file_name[STM_CM_archive_name_length],
   char error_message[STM_CM_message_length]
);
```

STM_CM_put_file

Checks in a file to the archive.

Syntax:

```
STM_CM_put_file
```

Input	bank_dir
	workfile_name
	temp_file_to_ci
	ext
	user_name
	comment_str
	keep_lock
Output	

```
int
STM_CM_put_file(
   const char *bank_dir,
   const char *workfile_name,
   const char *temp_file_to_ci,
   const char *ext,
   const char *user_name,
   const char *comment_str,
   int keep_lock,
   char revision[STM_CM_version_length],
   char error_message[STM_CM_message_length]
);
```

STM_CM_rollback

This API is an alternative to STM_CM_unlock_file. It executes the unlock command on a file in the databank. The API is called only if the CM DLL does not include implementation for `STM_CM_unlock_file`. In case that the `STM_CM_unlock_file` is not implemented, the “Unlock” menu entry in the Rational Statemate Configuration menus are visible to the user.

This API is optional and may not be implemented in the DLL.

Syntax:

```
STM_CM_rollback
```

Input	bank_dir name text full_archive_name user_name version
Output	

```
int
STM_CM_rollback(
   const char *bank_dir,
   const char *name,
   const char *ext,
   const char *full_archive_name,
   const char *user_name,
   const char [STM_CM_version_length],
   char error_message[STM_CM_message_length]
);
```

STM_CM_standalone_lock_file

Syntax:

```
int
STM_CM_standalone_lock_file(
   const char *bank_dir,
   const char *name,
   const char *ext,
   const char *full_archive_name;
   const char *user_name,
   const char *version,
   int in_load_op,
   char error_message[STM_CM_message_length]
);
```

STM_CM_unlock_file

Executes the unlock command on a file in the databank.

Syntax:

```
STM_CM_unlock_file
```

Input	bank_dir name text full_archive_name user_name version

```
int
STM_CM_unlock_file(
   const char *bank_dir,
   const char *name,
   const char *text,
   const char *full_archive_name,
   const char *user_name,
   const char *version,
   char error_message[STM_CM_message_length]
);
```

Information Specific to PVCS

This provides restrictions and branching information specific to the PVCS interface module.

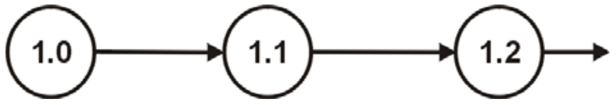
Restrictions

The following restrictions apply when using the PVCS interface module:

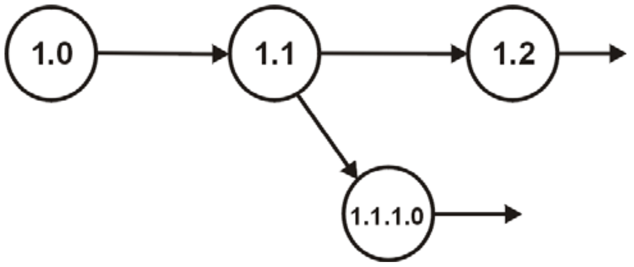
- PVCS does not control the workarea into which a file is placed when it is locked.
- PVCS "group" information has no meaning in the context of Statemate; therefore the access and permission fields in the Databank form also have no meaning when PVCS is used.

Branching

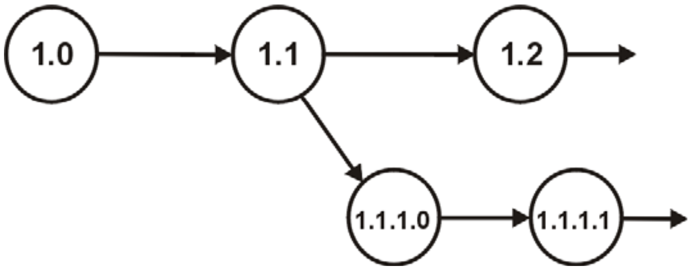
After checking out a configuration item with locking, the next check in puts the new revision as the next revision after the locked revision, rather than after the highest numbered revision. For example, take an item with three revisions:



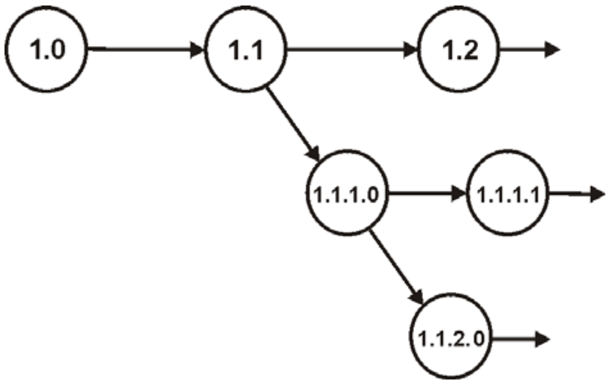
Checking out revision 1.1 and then checking it in creates revision 1.1.1.0, a branch.



Checking out 1.1.1.0 and checking it in creates revision 1.1.1.1., then 1.1.1.2 and so forth.



Checking out 1.1.1.0 again and checking in creates another new branch 1.1.2.0.



To return to the main branch, complete the following steps:

1. Release the lock.
2. Relock the item.
3. This locks the last revision on the main branch so the next check in is on the main branch.

Index