

Telelogic  
**StateMate®**

**Configuration Management**



**IBM®**



*Statemate*<sup>®</sup>

**Configuration Management**



Before using the information in this manual, be sure to read the “Notices” section of the Help or the PDF file available from **Help > List of Books**.

This edition applies to Telelogic Statemate 4.5 and to all subsequent releases and modifications until otherwise indicated in new editions.

© Copyright IBM Corporation 1997, 2008.

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

# Contents

---

<b>Configuration Management Tool</b> .....	<b>1</b>
<b>StateMate's CM Interface</b> .....	<b>2</b>
The Project Databank .....	2
Configuration Items .....	3
Locking .....	3
Version Numbers .....	4
Protection Levels and Groups .....	4
<b>The Databank</b> .....	<b>5</b>
Starting and Using the Databank .....	5
Databank Main Window .....	5
Configuration Item List .....	6
Item Properties .....	6
Existing Version List .....	7
<b>The Workarea</b> .....	<b>7</b>
<b>Configuration Management Operations</b> .....	<b>8</b>
<b>Using Configuration Management</b> .....	<b>10</b>
<b>Selecting a CM Tool for a StateMate Project</b> .....	<b>10</b>
<b>Defining Third-Party CM Tool Interfaces</b> .....	<b>13</b>
<b>Script-Based Interface to CM Tools</b> .....	<b>13</b>
How StateMate Determines Available CM Tools .....	15
Mapping Individual CM Operations to Scripts .....	16
Creating CM Operation Scripts .....	17
Parameter Types .....	18
<b>CM Script Reference</b> .....	<b>19</b>
calc_archive_name .....	19
delete_from_bank .....	19
delete_revision .....	20
get_file .....	20
get_locked_by .....	22
get_pattern .....	22
get_ver_from_bank .....	23

---

get_versions .....	23
init .....	24
is_file_in_bank .....	25
is_locked_in_bank .....	25
lock_file .....	25
modify_to_archive .....	26
put_file .....	26
unlock_file .....	27
<b>User Function Interface to CM Tool .....</b>	<b>28</b>
STM_CM_begin_databank_operation .....	30
STM_CM_calc_archive_name .....	30
STM_CM_close .....	30
STM_CM_delete_from_bank .....	31
STM_CM_delete_revision .....	32
STM_CM_end_databank_operation .....	32
STM_CM_get_file .....	33
STM_CM_get_files_list .....	34
STM_CM_get_file_last_modified_date .....	34
STM_CM_get_locked_by .....	35
STM_CM_get_pattern .....	35
STM_CM_get_ver_from_bank .....	36
STM_CM_get_versions .....	37
STM_CM_init .....	38
STM_CM_init_item .....	38
STM_CM_is_file_in_bank .....	39
STM_CM_is_locked_in_bank .....	39
STM_CM_lock_file .....	40
STM_CM_modify_to_archive .....	41
STM_CM_put_file .....	42
STM_CM_rollback .....	43
STM_CM_standalone_lock_file .....	44
STM_CM_unlock_file .....	44
<b>Information Specific to PVCS .....</b>	<b>45</b>
<b>Restrictions .....</b>	<b>45</b>
<b>Branching .....</b>	<b>45</b>

# Configuration Management Tool

---

In addition to its other features, Statemate provides built-in “checkin-checkout” capabilities similar to those found in products designed specifically for the task of configuration management (sometimes called change management, revision management, source control, and so forth).

Previously, Statemate used only its own Configuration Management (CM) tool. Now, you can transparently substitute the revision engine and repository format of a third-party configuration management tool for those of Statemate’s built-in facility. Add-on interface modules are available for the following widely-used CM tools:

Interface modules are available for the following widely-used, third-party configuration management tools. The following lists the supported CM tools and platforms:

- ◆ Continuous Software Corp.’s Continuous/CM
- ◆ Intersolv’s 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 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 Statemate's interface to a third-party CM tool, you cannot use the tool's own user 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, however, use the tool's own user interface for read-only operations such as viewing, searching, and reporting.
- ◆ Statemate does not support checking out different revisions of the same file. Only one revision can be checked out at a time.

## Statemate's CM Interface

Regardless of which CM engine and repository you use, the configuration management operations in Statemate's 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 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 Statemate.

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

## The Project Databank

Each 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

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, StateMate assigns it a version number by incrementing the highest existing version number.

StateMate's built-in CM tool tracks versions using whole numbers (positive integers). When you create a new item and check it into the databank, StateMate assigns it a version number of "1". If a third-party CM tool uses a different system of version numbering, the version numbers displayed by 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

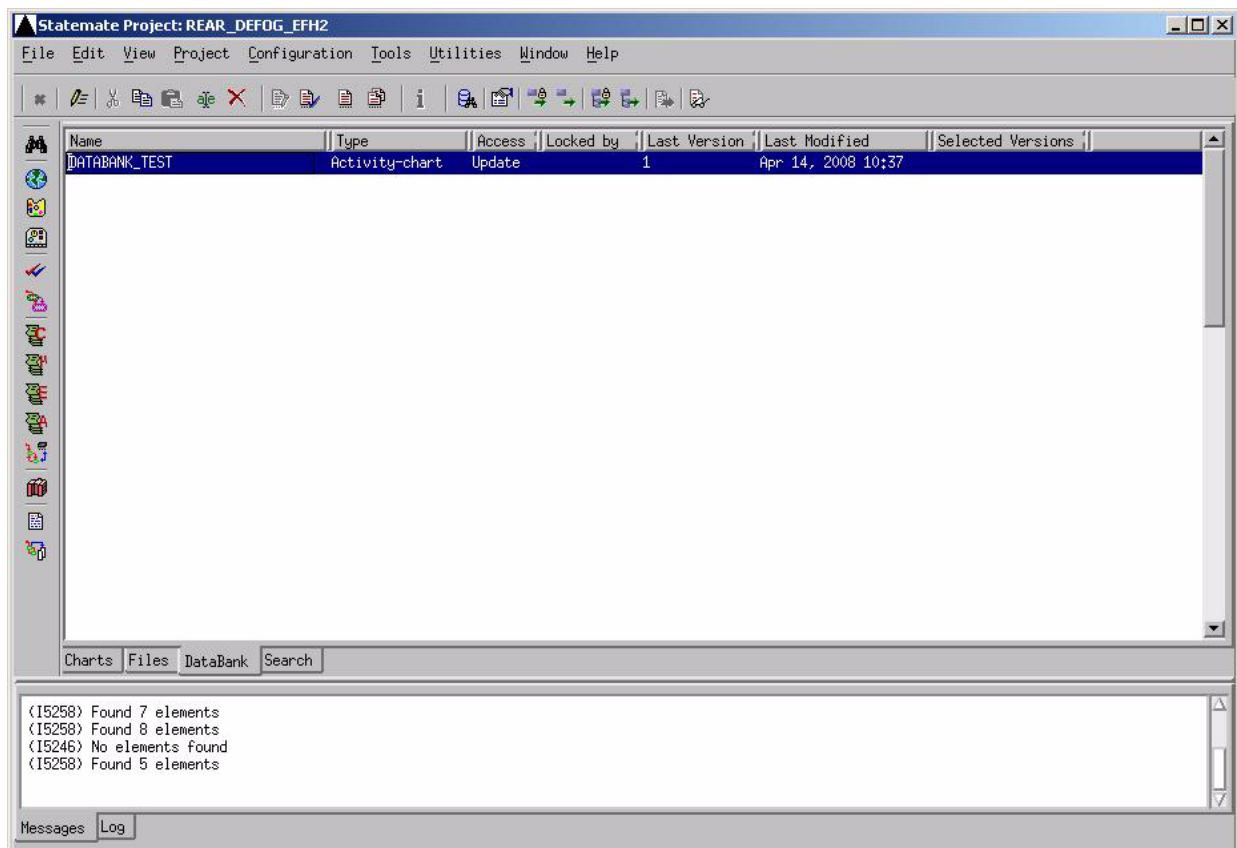
The Databank allows you to graphically navigate through the information in your databank.

## 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.



### Configuration Item List

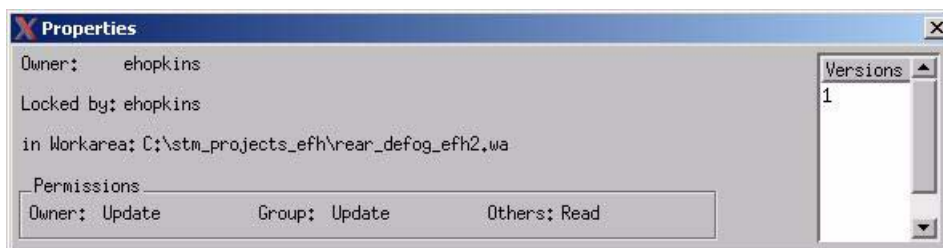
The Configuration Item List displays a list of the configuration items in the databank, as specified in the Show Files dialog:

- ◆ 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.



## Existing Version List

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

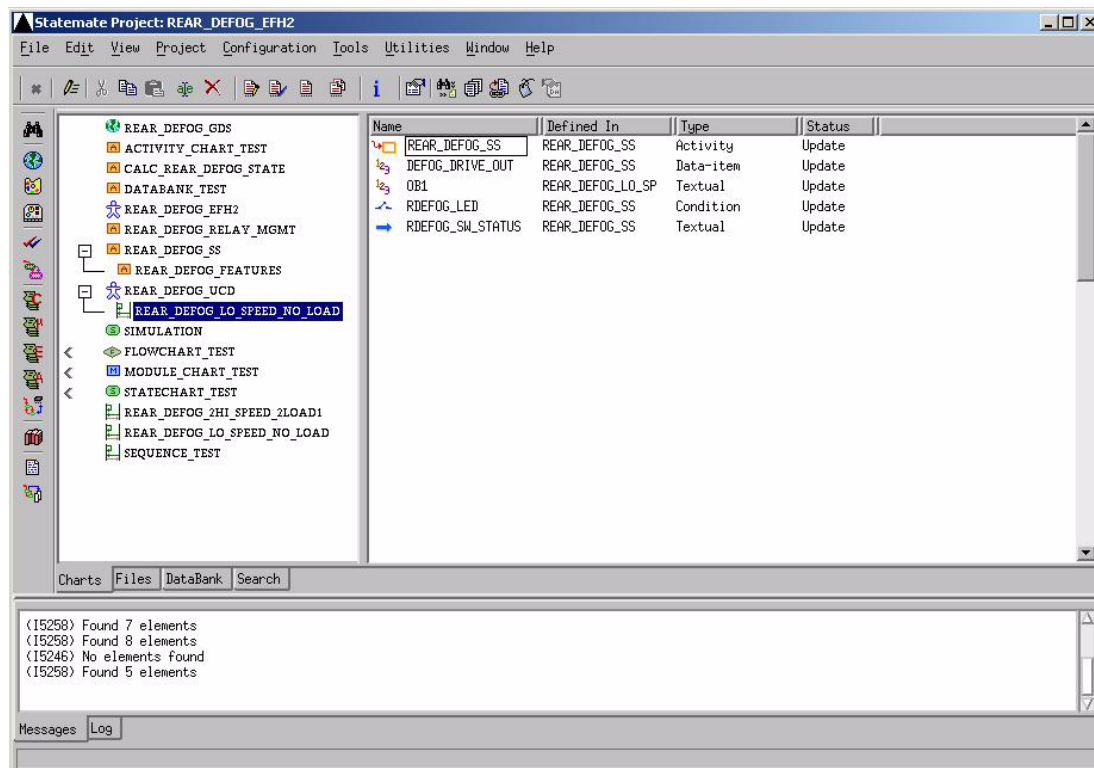
## The Workarea

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.



## Configuration Management Operations

Most of Statemate's 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
<b>Check In to Databank</b>	No	Yes	Saves the current version of the selected configurable items in the databank and updates their current revision numbers. Options are: <ul style="list-style-type: none"> <li>• Hold &amp; Keep Lock</li> <li>• Hold &amp; Release Lock</li> <li>• Check in &amp; Delete</li> </ul>
<b>Check In to Databank with Descendants</b>	No	Yes	Saves the current version of the selected configurable items and descendants in the databank and updates their current revision numbers. Options are: <ul style="list-style-type: none"> <li>• Hold &amp; Keep Lock</li> <li>• Hold &amp; Release Lock</li> <li>• Check in &amp; Delete</li> </ul>
<b>Check Out</b> <b>Check Out Chart/File</b>	Yes	Yes	Copies and locks (if 'with lock' is selected) files/charts from the databank to your workarea (update mode). Options are: <ul style="list-style-type: none"> <li>• With Lock</li> <li>• Without Lock</li> </ul>
<b>Check Out With Descendants</b>	Yes	Yes	Copies and locks (if 'with lock' is selected) files/charts and descendants from the databank to your workarea (update mode). Options are: <ul style="list-style-type: none"> <li>• With Lock</li> <li>• Without Lock</li> </ul>
<b>Update Workarea</b>	No	Yes	Opens Update Workarea dialog box.

Menu Operation	Databank	Workarea	Description
<b>Advanced Update Workarea</b>	No	Yes	Options are: <ul style="list-style-type: none"> <li>• Remove Unused Elements</li> <li>• Filtered Check Out from Databank of Selected</li> <li>• Filtered Check Out from Databank of All</li> <li>• Filtered Check Out of GDSs of Selected Charts</li> <li>• Filtered Check Out Parent from Databank</li> </ul>
<b>Check Out Parent from Databank</b>	Yes	Yes	Copies parent charts from the databank to your workarea
<b>Release Lock</b>	Yes	Yes	Unlocks files/charts in the databank that currently exist in your workarea (set read-only mode).
<b>Release Lock Notification</b>	Yes	No	Notifies you when a lock, set by another user on a chart/file, is released
<b>Lock</b>	Yes	No	Locks files/charts in the databank that currently exist in your workarea (sets update mode).
<b>Purge</b>	Yes	No	Deletes older versions of files/charts in the databank that you own.
<b>Delete</b>	Yes	No	Deletes files/charts in the databank that you own.
<b>Create Configuration</b>	No	Yes	Opens the Create Configuration dialog box.
<b>Execute Configuration</b>	Yes	Yes	Checks-out all charts/files in the configuration file into the workarea.
<b>Create/Modify Component</b>	No	Yes	Allows you to set up a component from a selected generic chart or an existing component.
<b>Remove Component</b>	No	Yes	Allows you to remove a component from the workarea.
<b>Component Versions</b>	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 Statemate release.

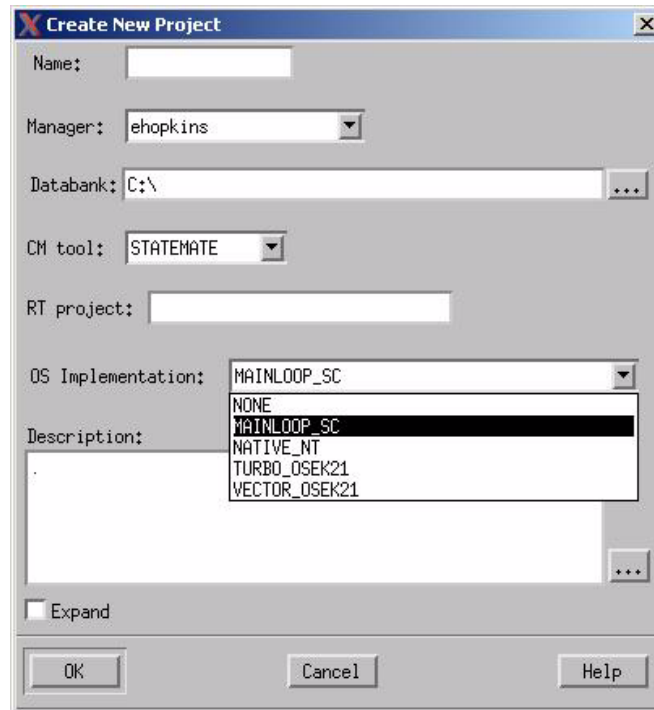
Menu Operation	Databank	Workarea	Description
<b>Create Configuration</b>	No	Yes	Creates a Configuration file that captures the current workarea. You can include all files or only selected files.  At the same time, you can check in the new Configuration file and/or the files that are in the configuration itself.
<b>Execute Configuration</b>	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 Statemate Project

The decision to use Statemate’s built-in facility or a third-party configuration management tool must be made when you create a 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 in the following figure.





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 Statemate interface to any third-party CM tool.

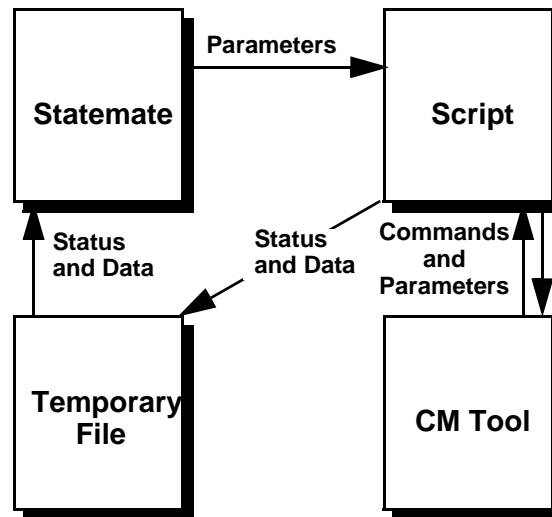
## Note

When using Statemate's interface to a third-party CM tool, you cannot use the tool's own user 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's own user 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 Statemate's Databank and Workarea are carried out by external executable files. Statemate itself does not directly communicate with your CM tool. Instead, it provides a generic CM interface. You provide the files that map each operation in Statemate's 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 shown in the following figure:



Since 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 Statemate's CM operations to your scripts, as explained in the sections that follow.

## How Statemate Determines Available CM Tools

In 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 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 Statemate to a script that actually performs the operation. The contents of the mapping file template are shown in the following table:

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

#### Note

---

The list of operations in this file is defined by 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 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 Statemate by creating a temporary file. Statemate deletes the temporary file after reading it.

## Parameter Types

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

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

**Description:**

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

**Description:**

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

### Description:

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
<hr/>	
output	ret_val

## get\_file

### Description:

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
<hr/>	
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

### Description:

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

### Description:

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

### Description:

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

### Description:

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

#### Description:

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

### Description:

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

### Description:

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

### Description:

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

### Description:

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

### Description:

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

### Description:

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
<hr/>	
output	ret_val

## User Function Interface to CM Tool

There is an API in the 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

### Description:

Starts the databank operation.

### Syntax:

```
int
STM_CM_begin_databank_operations(
    char_error_messages[STM_CM_message_length])
```

## STM\_CM\_calc\_archive\_name

### Description:

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

### Description:

Closes the databank.

### Syntax:

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

## STM\_CM\_delete\_from\_bank

### Description:

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

### Description:

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

### Description:

Ends the databank operation.

### Syntax:

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

## STM\_CM\_get\_file

### Description:

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

### Description:

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 then 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

### Description:

Retrieve the last modification date of chart/file

### Syntax:

```
int
STM_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_message[STM_CM_message_length]
);
```



## STM\_CM\_get\_locked\_by

### Description:

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_get_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

### Description:

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

### Description:

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

### Description:

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 returns 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

### Description:

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

### Description:

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

### Description:

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

### Description:

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
Output	

```
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

### Description:

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

### Description:

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

### Description:

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

### Description:

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 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

### Description:

### 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

### Description:

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:

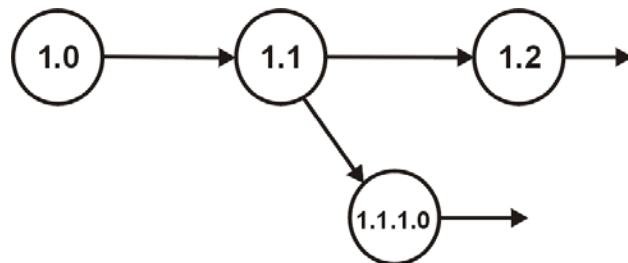
- ◆ PVCS does not control the workarea into which a file is placed when it is locked.
- ◆ PVCS “group” information has no meaning in the context of Statemate; therefore the access and permission fields in the Databank form also have no meaning when PVCS is used.

## Branching

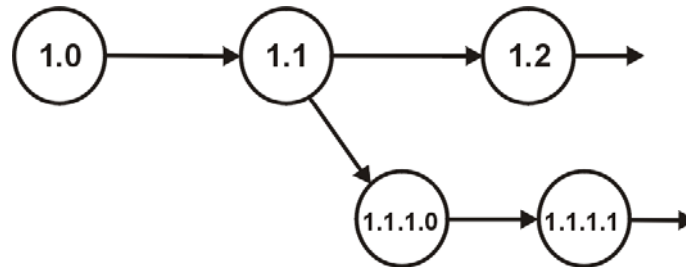
After checking out a configuration item with locking, the next check in puts the new revision as the next revision after the locked revision, rather than after the highest numbered revision. For example, take an item with three revisions:



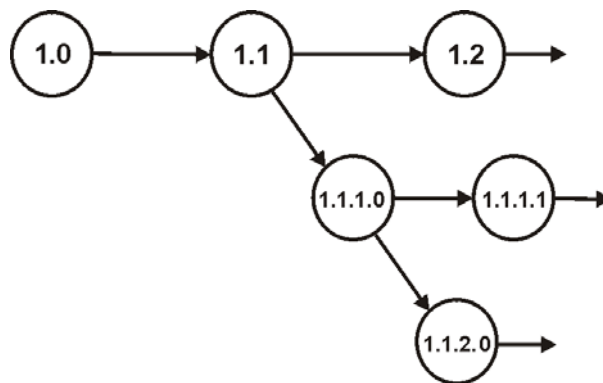
Checking out revision 1.1 and then checking it in creates revision 1.1.1.0, a branch.



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

---

## B

bank\_dir 18  
Branching, PVCS 45

## C

ClearCase 1  
CM operations  
  mapping to scripts 16  
CM tool  
  adding to list 15  
  availability 15  
  defining new interface 16  
  script-based interface 13  
  selecting default 11  
CM\_USER\_FUNC\_calc\_archive\_name 16  
CM\_USER\_FUNC\_delete\_from\_bank 16  
CM\_USER\_FUNC\_delete\_revision 16  
CM\_USER\_FUNC\_get\_file 16  
CM\_USER\_FUNC\_get\_locked\_by 16  
CM\_USER\_FUNC\_get\_pattern 16  
CM\_USER\_FUNC\_get\_ver\_from\_bank 16  
CM\_USER\_FUNC\_get\_versions 16  
CM\_USER\_FUNC\_init 16  
CM\_USER\_FUNC\_is\_file\_in\_bank 16  
CM\_USER\_FUNC\_is\_locked\_in\_bank 16  
CM\_USER\_FUNC\_lock\_file 16  
CM\_USER\_FUNC\_modify to archive 16  
CM\_USER\_FUNC\_put\_file 16  
CM\_USER\_FUNC\_unlock\_file 16  
Configuration item list 6  
Configuration items, definition 3  
Configuration management 10  
  operations 8  
Configuration management operations  
  Configuration menu 8  
Configuration menu operations 8  
Continuus 1  
Creating scripts 17

## D

Databank 2, 5  
  main window 5  
  starting 5  
delete\_from\_bank 19  
delete\_revision 20  
Dynamic-library 28

## E

Existing version list 7  
ext 18

## F

full\_archive\_name 18

## G

get\_file 20  
get\_locked\_by 22  
get\_pattern 22  
get\_ver\_from\_bank 23  
get\_versions 23

## I

init 24  
Interface, defining new 16  
Intersolv 1  
is\_file\_in\_bank 25  
is\_locked\_in\_bank 25  
Item properties 6

## L

lock 18  
lock\_file 25  
Locking 3

### M

Main branch, returning to 46  
Mapping CM operations to scripts 16  
Mapping file, definition 16  
modify\_to\_archive 26

### N

name 18

### P

Paramter types 18  
Projects  
  databank 2  
  selecting CM tool 10  
Protection groups 4  
Protection levels 4  
put\_file 26  
PVCS 1  
  branching 45  
  interface module 45  
  restrictions 45

### R

Rational Software, ClearCase 1  
ret\_val 18  
revision 18

### S

Scripts, creating 17  
STM\_CM\_begin\_databank\_operation 30  
STM\_CM\_calc\_archive\_name 30  
STM\_CM\_close 30  
STM\_CM\_delete\_from\_bank 31  
STM\_CM\_delete\_revision 32

STM\_CM\_end\_databank\_operation 32  
STM\_CM\_get\_file 33  
STM\_CM\_get\_file\_last\_modified\_date 34  
STM\_CM\_get\_files\_list 34  
STM\_CM\_get\_locked\_by 35  
STM\_CM\_get\_pattern 35  
STM\_CM\_get\_ver\_from\_bank 36  
STM\_CM\_get\_versions 37  
STM\_CM\_init 38  
STM\_CM\_init\_item 28, 38  
STM\_CM\_is\_file\_in\_bank 39  
STM\_CM\_is\_locked\_in\_bank 39  
STM\_CM\_lock\_file 40  
STM\_CM\_modify\_to\_archive 41  
STM\_CM\_put\_file 42  
STM\_CM\_rollback 43  
STM\_CM\_unlock\_file 44

### T

temp\_result\_file 18  
tmp\_file\_to\_ci 18

### U

unlock\_file 27

### V

Version numbers 4

### W

Workarea 7  
Workarea, definition 3  
workarea\_dir 18  
workfile\_name 18  
working\_dir 18