Chapter 13

13 Command Line Interface

SpectrumSCM supports a Command Line Interface so that tasks can be performed over ASCII terminals and reports and builds can be performed using scripts. Command Line commands can be run from the Unix or Linux command line, an xterm window, the Windows Command Prompt Screen, or used within a script.

To use to the Command Line interface in a Windows environment, start the Command Prompt (cmd.exe) via Start / Programs/ Accessories / Command Prompt. In UNIX or Linux, use the command line or an xterm window.
Commands are executed from the <SCMUi install> bin directory:

```
Unix or Linux example:    > cd   /home/user/scm/bin
Windows example:          > cd   C:\home\user\scm\bin
```

To access the Command Line interface there are 3 security options, these basically cover how the user is verified and allowed access to the repository server and information. The command-line infrastructure defaults to use the users current operating system user-id, the "-login" option can be specified to change this. The password can then be provided via the "-password" option, supplied from the terminal using the "-prompt" option or thirdly via accessControl. AccessControl basically states that user X on workstation Y is a pre-verified (or trusted) user and therefore needs no further password (this is also sometimes called single sign-on, where a user has to logon to their workstation but then that information is passed to the applications as their sign-on information). AccessControl can be enabled through the Server Configuration Wizard. (*See details in Chapter 12, Unauthenticated Commandline Access*).

All the command line functions display detailed usage statements to the standard output when the command is specified without parameters.

Command	Usage
scm_addnote	Add a note to the supplied CR
scm_bulkassign	Bulk assign a set of CRs from one situation to another
scm_co	Check out a file for read.
scm_cow	Check out a file for edit.
scm_check	Check what we currently have out for edit.
scm_ci	Check in a file.
scm_cidir	Check in a set of files.
scm_diff	Initiate the standalone UI file or directory differencer
scm_getUserProjectInfo	Extract an XML statement of the specified users project view.
scm_gv	Extract the current version of the product or directory.
scm_mkdirs	Create the relative workspace directories.
scm_gcr	Extract only the files touched under the specified change request
scm_gr	Extract the specified release of the product.

scm_gir	Extract the intermediate release from the specified life-cycle phase
scm_gpack	Extract the specified package and its components.
scm_mngeMeta	Manage a files meta-information/notes.
scm_newcr	Create a new Change Request.
scm_newXMLcr	Create a new Change Request using an XML template
scm_progress	Progress the state of a worked Change Request.
scm_rpt	Reports interface.
scm_unlock	Unlock a file from edit.
scm_updateXMLcr	Update the specified CR via an XML template

13.1 Parameters

All command line arguments require some set of these parameters. Some arguments are mandatory; others are optional.

<u>**NOTE:**</u> If an argument value contains embedded spaces, the value must be quoted.

13.2 Commonly used parameters that require an argument.

If a command uses one or more among these parameters, an argument is required.

Parameter	Argument and Usage
-root <root directory>	This is the current local root directory for the project. i.e. the root of the disk location where operations should be performed.
-project <project Name>	The project name
-generic <generic >	The generic name
-filename <filename>	The file name
-release <release name>	The release name
-cr <cr>	The CR associated with an action. For example, if the user wishes to edit the file, the -edit argument and the associated CR should be specified
-host <server>	Specify the server, if you are not on the system containing the SpectrumSCM server
-port <port number>	If your server is not using the default (1099) port, specify the port number here.
-proxyhost <host>	If the proxy feature is being used, where is the proxy located
-proxyport <port>	If the proxy feature is being used, which port is it using on the host.
-login <login id>	your login id.
-password <password>	This option can be used to directly provide your server login password.

13.3 Parameters without Arguments

If any of these parameters are specified for a command, the parameters do not require arguments. Using the parameter with a command will result in the described action. .

Parameter	Action
-dirsOnly	Used with commands scm_gv and scm_mkdirs, this will either extract or create the directory structure for the project.
-binary | -text | -ascii	Used with the command SCM_CI to specify the type of file being check in. ***NOTE:*** -text and -ascii are synonymous. If no argument is specified for check in, system will determine file type automatically. If there are any characters outside of the normal ASCII range of printable characters, the file type is assumed to be Binary
-includeBinaries	Use to indicate that binaries are to be included
-overwrite	Use if the file is to be overwritten
-recurse	This argument will walk the entire directory tree from the point specified and perform the operation requested.
-ssl	Enable secure socket layer interface ***NOTE:*** The –ssl option cannot be used unless the secure socket layer (ssl), a security option, is enabled on the server. Security options must be enabled by an authorized administrator.
-edit	Used only in scm_co to flag a file to be cheeked out for edit.
-prompt	Requests that the server prompt for the users password.

13.4 Environmental Variables

Two environment variables are supported to ease use, allowing presets for Java Virtual Machine memory and common arguments such as server connectivity and project information. Setting environment variables at the beginning of a command line session or in a script will eliminate the need to set the parameters in each of the subsequent command line commands. This increases productivity and reduces the chance for typing errors.

SCM_CMDL_ARGS

Use SCM_CMDL_ARGS to reduce typing with respect to common command line options.

For example, setting SCM_CMDL_ARGS to "-project Genesis -generic gen1.0" will automatically supply those arguments to each command line function invocation without repeated typing.

Using the SCM_CMDL_ARGS option, this example sets the project name to Genesis and the generic name to gen1.0 for all subsequent commands and thus they need not be repeated. This increases productivity and reduces the chance for typing errors. The parameters are set once for the entire script or session.

```
$ export SCM_CMDL_ARGS="-project Genesis -generic gen1.0 –root /home/user/gen10"
. . .
$ scm_co  -filename Test.java
$ scm_cow -filename Test.java -cr TestPrj000005
$ scm_ci –filename Test.java –cr TestPrj000005
$ scm_gv
$ scm_gr –release baseline
```

Depending on your security settings, you could also specify your login and password (or prompt) options here too.

SCM_JAVA_ARGS

SCM_JAVA_ARGS are used to fine-tune the Java Virtual Machine.
Setting SCM_JAVA_ARGS to "-Xmx128m" can be used to control the server memory allowance available to the Java Virtual Machine (setting it to 128MB in this case). The default heap size for the VM is 64MB.

Using the SCM_JAVA_ARGS option, an example is:

```
$ export SCM_JAVA_ARGS=”-Xmx128m”
. . .
$ scm_gv -root /home/user/scm -project Genesis -generic gen1.0
```

Setting $ export SCM_JAVA_ARGS=”-Xmx128m” could be used when the user needs more memory when executing an extremely large report or extracting a very large directory structure.
Error messages such as "out of memory" or "stack overflow" indicate the need for more memory.

13.5 Commands

Command line commands can be run from the command line (or Windows Command Prompt Screen) or used within a script. **For help in the command line interface,** type in the command. All of the command line functions display detailed usage statements to the standard output if only the command name is input.

Command	Arguments
scm_co Check out a file	-root <root> -project <project> -generic <generic> (-filename <filename> | -filelist <filelist file> | -bomfile <BOM file>) [-edit] if the file is being checked out for edit [-cr <CR number>] [-overwrite] if the file needs to be overwritten [-common] [-uncommon] [-version <version>] [-host <SCM host>] [-port <SCM port>] [-proxyhost <proxy host>] [-proxyport <proxy port>] [-login <login>] [-password <password> | -prompt] [-ssl]

Example: To check out a file Test.java from the source directory src/java under a user's SCM directory /home/user/scm; the current directory should be /home/user/scm/src/java where the file Test.java resides.
The user should execute either:

```
$ scm_co -root /home/user/scm -filename Test.java -project Genesis -generic gen1.0
```

(to extract a read-only copy of the head revision of Test.java)

```
$ scm_co -root /home/user/scm -filename Test.java -project Genesis -generic gen1.0 -edit -cr
TestPrj000005
```

(to extract for edit the Test.java file)

NOTE: If the user wishes to edit a file, either the **scm_cow** command or the scm_co command can be used with the -edit argument and the appropriate CR specified.

The ***filename*** option is the most frequently used. However, if you wish to check-out a significant number of files, place that list of files (one per line) into a temporary file. Then use the ***filelist*** option and supply that temporary file.

The ***bomfile*** option allows you to specify a Bill Of Materials report generated from a release extract (GUI or command-line, intermediate or full release). The scm_co operation will then fully reproduce that initial extract by pulling the specific file versions from the repository. Use the HTML BOM report here as opposed to the textual file.

Command	Arguments
scm_cow Check out a file for edit.	-root <root> -project <project> -generic <generic> -filename <filename> -cr <cr> [-overwrite] [-common] [-uncommon] [-host <SCM host>] [-port <SCM port>] [-login <login>] [-ssl]

Example: To check out a file Test.java from the source directory src/java under a user's SCM directory /home/user/scm;

Change directory to /home/user/scm/src/java where the file Test.java resides.
Execute:

```
$ scm_cow -root /home/user/scm -filename Test.java -project Genesis -generic gen1.0
```

Command	Arguments
scm_check Check what a user currently has out for edit. Default user is the user currently logged into SCM. Use –user argument to see what another user has checked out.	-project <project> [-user <userid>] [-host <host>] [-port <port>] [-ssl]

Example: To view a list of files that a user has checked out for the project Genesis, this command should be used.

```
$ scm_check -project Genesis
                                                      2002/11/21 08:34:39
                               User Edited Files
                              ------------------
--------------------------
| Project :  | Genesis    |
| User :     | rich       |
--------------------------
-----------------------------------------------------------------------------------------
|File Name    |Change Request| Generic | Version No | Edited on           | Creator      |
                                                                                        |
|install.java | TestPrj000005| gen1.0  | 1.39       | 2002/05/19 08:56:19 | Gene         |
                                                                                        |
-----------------------------------------------------------------------------------------
End Of Report
```

Command	Arguments
scm_ci Check in a file.	-root <root> -project <project> -generic <generic> (-filename <filename> | -filelist <filelist file>) -cr <cr> [-binary | -text | -ascii] ***Note*** *-text and -ascii are synonymous, if none are specified Checkin will determine file type automatically.* [-host <SCM host>] [-port <SCM port>] [-login <login>] [-ssl]

Example: To check in a file Test.java to the source directory src/java under a user's SCM directory /home/user/scm;

Change directory to /home/user/scm/src/java where the file Test.java resides.

Execute:

```
$ scm_ci –root /home/user/scm -filename Test.java -cr TestPrj000005 -project Genesis -generic gen1.0
```

Command	Arguments
scm_cidir Check in a set of files contained under the current working directory.	-root <root> -project <project> -generic <generic> -cr <cr> [-host <SCM host>] [-port <SCM port>] [-login <login>] [-recurse] [-includeBinaries] [-ssl]

Example: To check in a directory to the source directory src/java under a user's SCM directory /home/user/scm;

Change directory to /home/user/scm/src/java where the target files reside.

Execute:

```
$ scm_cidir -root  /home/user/scm -cr TestPrj000005 -project Genesis -generic gen1.0
```

Command	Arguments
scm_gv Extract the current version of the product (get version)	-root <root> -project <project> -generic <generic> [-recurse] [-dirsOnly] [-host <SCM host>] [-port <SCM port>] [-login <login>] [-ssl]

Example: To obtain the contents of the src/java directory under the user's current SCM directory /home/user/scm, the user must:

Change directory to the /home/user/scm/src/java directory.

Execute:

```
$ scm_gv –root /home/user/scm -project Genesis -generic gen1.0
```

Command	Arguments
scm_mkdirs Create the relative workspace directories (Note – this is really just a wrapper for scm_gv –dirsOnly).	-root <root> -project <project> -generic <generic> [-recurse] [-host <SCM host>] [-port <SCM port>] [-login <login>] [-ssl]

NOTE: If the user wishes to obtain subdirectories, the -recurse argument should be appended.

Example: To obtain the contents of the src/java directory under the user's current SCM directory /home/user/scm, the user must:

Change directory to the /home/user/scm/src/java directory.

Execute:

```
$ scm_mkdirs –root  /home/user/scm -project Genesis -generic gen1.0
```

Command	Arguments
scm_gr Extract the specified release of the product.	-root <root> -project <project> -generic <generic> -release <release> [-host <SCM host>] [-port <SCM port>] [-login <login>] [-ssl]

Example: To extract an entire release, to perform a release build for example, you can run this from the command line or from inside an automated build script.

```
$ scm_gr –root /home/user/scm -project Genesis -generic gen1.0 -release scm2.6
```

Command	Arguments
scm_gir Extract the specified interim release of the product based on all the CRs at or past the specified phase.	-root <root> -project <project> -generic <generic> -phase <phase> [-excludeBase] [-includePhase <comma-list>] [-excludePhase <comma-list>] [-nodeps] [-deltas] [-host <SCM host>] [-port <SCM port>] [-login <login>] [-ssl]

Example: To extract an intermediate release build for all issues ready for integration testing, you can run this from the command line or from inside an automated build script.

```
$ scm_gir –root /home/user/scm -project Genesis -generic gen1.0 –phase “Integration Test”
```

The ***nodeps*** option will turn OFF dependancy checking, this means that ALL CRs in the specified phase and later will be extracted. With dependancy checking ON (the default and safest option), only CRs that do not depend on other incomplete CRs will be included.

The ***deltas*** option will only extract files that have changed since the last *gir* command. This is controlled by the Bill Of Materials file that is produced into the root directory with each *gir* execution.

The ***excludeBase*** option will only extract the file versions relative to the non-released CRs i.e. only extract those files that are still being worked.

The ***includePhase*** and ***excludePhase*** lists are not normally needed as the interim release mechanism will automatically pick up the appropriate life-cycle phases based on the linear project life-cycle. However, in the case of an extensive graphical (non-linear) workflow specific phases might want to be included or excluded if the linear order is not quite correct for the desired extract.

Command	Arguments
scm_gcr Extract the set of source file versions that were edited by this CR.	-root <root> -project <project> -generic <generic> -cr <cr> [-before] [-createBOM | -bomOnly] [-host <SCM host>] [-port <SCM port>] [-login <login>] [-ssl]

Example: To extract just the files touched by CR TestPrj000004.

```
$ scm_gcr –root /home/user/scm -project Genesis -generic gen1.0 -cr TestPrj000004
```

The ***createBOM*** option will create a Bill of Materials report in addition to performing the file version extracts. The ***bomOnly*** option will only create the Bill of Materials report and will not extract the file versions.

The ***before*** option will extract (or report) what file versions existed in the repository BEFORE this CR. I.E. In a simple linear case where this CR edited version 1.3 of file X, then the –before option will extract version 1.2 of file X. This option is useful in conjunction with code review type procedures or tools to state exactly what the before and after situations were.

Command	Arguments
scm_unlock Unlock a file from edit.	-root <root> [-host <SCM host>] [-port <SCM port>] -project <project> -generic <generic> -filename <filename> [-ssl]

Example: To unlock a file that was checked out and locked, execute:

```
$ scm_unlock -root /home/user/scm -filename Test.java -cr TestPrj000005 -project Genesis \
  -generic gen1.0
```

Command	Arguments
scm_newcr Create a new Change Request.	[-project <project>] [-generic <generic>] [-header <header>] [-description<description>] [-host <SCM host>] [-port <SCM port>] [-login <login>] [-ssl]

Example: To create a new CR,

```
$ scm_newcr
Project: scm_utils
CR creation Phases:
        (1) Study
        (2) Develop
        (3) Test
Please select a CR creation phase: 1
Please select one value from each presented attribute.
"Severity"  values:
        (1) High
        (2) Medium
        (3) Low
Please select a value: 2
Attribute value selection completed.
Please provide a brief description of this CR:
Test of command line cr create
Please provide a detailed description of this CR; terminate the description
with a single period '.' on a line:
Test of command line cr create
.
Would you like to have this CR assigned to you? (y/[n]): y
Generics:
        (1) base
Please select a generic: 1
Assignment Phases:
        (1) Study
        (2) Develop
        (3) Test
Please select a phase: 1
You have provided the following information:
1) Project:           scm_utils
2) Create State:        Study
3) Attributes:
        Severity       Medium
4) Header:              Test of command line cr create
5) Description:
Test of command line cr create
6) Assigned User:       rich
   Assigned Phase:      Study
   Generic:             base
Are these correct? If so, enter <y>, otherwise enter the number of the incorrect field: y

CR scm_utils000014 successfully created.
$
```

Command	Arguments
scm_newXMLcr Create a new Change Request using an XML template.	[-project <project>] [-generic <generic>] [-host <SCM host>] [-port <SCM port>] [-login <login>] [-file <template>]

Example: To create a new CR,

```
$ scm_newXMLcr –project Genesis – generic Mainline –file c:\Work\template.xml –login scm
```

A sample XML template is shown below:

```
<CRS>
      <NEWCR>
            <HEADER     VALUE="This is the header"/>
            <DESCRIPTION>
                  <DESCITEM VALUE="Now is the time for"/>
                  <DESCITEM VALUE="all good programmers"/>
                  <DESCITEM VALUE="to come to the aid of"/>
                  <DESCITEM VALUE="their editors"/>
            </DESCRIPTION>
            <CREATOR VALUE="scm"/>
            <ATTRIBUTES>
                  <ATTRIBUTE_ENTRY NAME="Severity" VALUE="High"/>
                  <ATTRIBUTE_ENTRY NAME="Location" VALUE="Atlanta"/>
            </ATTRIBUTES>
            <CREATION_PHASE VALUE="Study"/>
            <ASSIGNED_USER>
                  <USER VALUE="scm"/>
                  <PHASE VALUE="Study"/>
                  <GENERIC VALUE="Mainline"/>
            </ASSIGNED_USER>
            <HISTORY>
                  <USER VALUE="scm"/>
                  <HISTORY_ENTRY VALUE="some info"/>
                  <HISTORY_ENTRY VALUE="some info"/>
            </HISTORY>
      </NEWCR>
</CRS>
```

Command	Arguments
scm_addnote Add a note to a Change Request.	[-project <project>] [-cr <Change Request No>] [-host <SCM host>] [-port <SCM port>] [-login <login>] [-ssl]

NOTE: This command will prompt for all necessary arguments.

Example: The entered information is in bold.:

```
$ scm_addnote –project SD_Demo – cr SDCR000031 –login scm

Please provide the note to be added to this change request.
Terminate the note with a single period '.' on a line
This is my note
.

You have provided the following information:
1) Project:             SD_Demo
2) Change Request:  SDCR000031
3) Note:
This is my note

Are these correct? If so, enter <y>, otherwise enter the number of the
incorrect field: y

CR SDCR000031 successfully annotated.
$
```

Command	Arguments
scm_mngeMeta Get, Set or Append meta-information to the specified file.	-project <project> -generic <generic> -root <root> -filename <filename> (-get | -set | -append) -meta "meta information" [-host <SCM host>] [-port <SCM port>] [-login <login>] [-ssl]

For example, to add meta-information to file Test.java from the source directory src/java under a user's SCM directory /home/user/scm; the current directory should be /home/user/scm/src/java where the file Test.java resides.

The user should execute:

```
$ scm_mngeMeta -root /home/user/scm -filename Test.java -project <project> \
      -generic <generic> -set -meta "new meta information"
```

Command	Arguments
scm_bulkassign Bulk assign a set of CRs based on specified criteria	[-project <project>] [-generic <generic>] [-cp <Current Phase>] [-ca <Current Assignee>] [-cr <Current Release>] [-np <New Phase>] [-na <New Assignee>] [-note <Annotation>] [-host <SCM host>] [-port <SCM port>] [-login <login>] [-ssl]

NOTE: This command will prompt for all necessary arguments.

Example: The entered information is in bold.:

```
$ scm_bulkassign -project SD_Demo -generic Mainline -cp Test -ca ken -cr xyz123 -np Release -
na eric -login scm

Please provide the note to be added to these change request(s).
Terminate the note with a single period '.' on a line
This is my note
.

You have provided the following information:
1) Project:            SD_Demo
2) Generic:             Mainline
3) Current Phase:       Test
4) Current Assignee:    ken
5) Current Release:     xyz123
6) New Phase:          Release
7) New Assignee:        eric
8) Note:
This is my note

Are these correct? If so, enter <y>, otherwise enter the number of the
incorrect field: y

Found CR SDCR000011
Found CR SDCR000017
Found CR SDCR000005
Found CR SDCR000007
Confirm re-assignment of 4 CRs (y/n) > y

CR SDCR000011 successfully assigned.
CR SDCR000017 successfully assigned.
CR SDCR000005 successfully assigned.
CR SDCR000007 successfully assigned.
$
```

Command	Arguments
scm_progress Progress the state of a worked Change Request.	[-project <project>] [-cr <Change Request No>] [-host <SCM host>] [-port <SCM port>] [-login <login>] [-ssl]

NOTE: This command will prompt for all necessary arguments.
Example: The entered information is in bold.:

```
$ scm_progress

Please enter the Project: scm_utils
Please enter the Change Request no: scm_utils000002
Please provide some description of the changes performed.
Terminate the description with a single period '.' on a line
test progress
.

You have provided the following information:
1) Project:           scm_utils
2) Change Request:     scm_utils000002
3) Description:    test progress

Are these correct? If so, enter <y>, otherwise enter the number of the
incorrect field:  y

CR scm_utils000002 successfully progressed.
$
```

Command	Arguments
scm_rpt Reports interface	<-xmlinputfile <filename> || -project <project name> [- template] [-host <host>] [-port <port>] [-login <login>] [-width <width>] [-ssl]

The command line report functionality operates in one of two modes, interactive or XML driven. In the interactive mode the user is prompted by the report mechanism to choose a particular report to run, and is then prompted for each of the arguments for the chosen report. Alternatively, the user can execute the command line report mechanism in an XML driven mode. In XML mode, the user supplies an XML template which describes the report to execute and all of the necessary input parameters.

To drive the system in interactive mode, the user should supply the "-project" option along with the necessary "-host" and "-login" options like the following example:

$ scm_rpt –project MYPROJECT –host MYSERVER –login MYLOGIN

The report functionality will respond with a list of all available reports, including custom reports and the user will be prompted to choose one of the reports. Once a report has been chosen, the user will be prompted for the specific input parameters for the chosen report.

Note that specific options can also be specified on the commandline as parameters. For example:
$ **scm_rpt –project MYPROJECT –host MYSERVER –login MYLOGIN –report 1 –cr_number TestPrj000005 –output_style text**
Which shows the CR report for TestPrj00005, and by using SCM_CMDL_ARGS or scripting typing can be reduced considerably.

To generate a template for use with the XML input option, use the "-template" option to signal for the creation of an XML template:
$ scm_rpt –project MYPROJECT –host MYSERVER –login MYLOGIN –template
After the user has chosen the report and added all of the arguments for the report the command line report mechanism will prompt the user for a file location. The user should choose a file location and name that corresponds to the name of the chosen report, i.e. **ChangeRequestReport.xml**.

The following is an example of a completed CR Assignee report input file in XML format.

```
<Root>
  <Argument>
    <Name val="Report_Name"/>
    <Input_Value val="scm.implementation.reports.XMLCrAssignHistory"/>
  </Argument>
  <Argument>
    <Name val="Project"/>
    <Input_Value val="SCM"/>
  </Argument>
  <Argument>
    <Name val="Assignee"/>
    <Input_Value val="scm"/>
  </Argument>
  <Argument>
    <Name val="Start Date (yyyy/mm/dd)"/>
    <Input_Value val="1990/01/01"/>
  </Argument>
  <Argument>
    <Name val="End Date (yyyy/mm/dd)"/>
    <Input_Value val="2003/04/30"/>
  </Argument>
  <Argument>
    <Name val="Output style"/>
    <Input_Value val="CSV"/>
  </Argument>
</Root>
```

This XML control file can then be passed to the scm_rpt commandline to execute the report and produce the appropriate report. Once the XML control file has been defined it can be re-used many times, for example to automatically generate a weekly status report. In addition, the user could use scripting languages like SED, AWK or Perl to modify the XML input file before it is processed by the command line report function to update date (or other) fields as desired. The following example illustrates how to run the report functionality via an XML input file:

```
$ scm_rpt –xmlinputfile MYFILE.xml –host MYHOST –login MYLOGIN
```

<u>**NOTE**</u>***: See Chapter 10, Reports for a complete description pre-defined reports and how to develop and save customized reports.***