Chapter 9

9 Release/Package Management

This chapter describes how files can be packaged to create a product release. You will learn how SpectrumSCM ensures the integrity of a release, its automatic CR dependency checks, and how to select CRs to create a release. Under SpectrumSCM version 2.6 the Package/Component Management feature has been introduced which allows product extract/builds to be assembled from multiple release/project components.

9.1 What is a Release?

A release is a set of files, each at a particular version, that when extracted from the system, make up a single version of the product. Managing release formation can be a tedious, time-consuming job on some CM systems. In order to create a release with separate versions of individual files, a CM system needs to be able to properly track the file changes that make up the individual file versions and present that information to the user in some meaningful form. In the **SpectrumSCM** system, this is easily accomplished with the built-in issue tracking system.

Specialized file changes, like small branches in the code, cannot be lost in the system because each change is officially recorded in the issue tracking system. Each release is a collection of active CRs in the system. Releases build on top of previous releases to simplify the entire release management job. As each CR is completed and the releases are created, the number of outstanding CRs that must be accounted for is reduced to the number of open CRs since the last release was created. This keeps the number of CRs for any given release down to a manageable few.

A release (a specific version of the product) can therefore be easily re-created at any time simply by extracting the relevant versions of the necessary files using the appropriate CRs. Outside of tracking changes to files, creating releases is the single most important job that a CM system must perform. The **SpectrumSCM™** system is designed from the ground up with the necessary features that make release management a simple task.

9.2 Releases and their Relationship to Generics

Releases are defined within a generic and a generic can contain a sequence of releases. The first release in a new generic is based on the last release in the predecessor generic. This is done in the **SpectrumSCM** system as a space saving measure and it keeps in line with the overall practice of basing one release on another.

NOTE: When a new generic is formed, the last release in the predecessor generic will become locked, and that release will be used as a basis for subsequent releases in the descendent generic.

9.3 The Release Management Process

By design, the **SpectrumSCM** system imposes no rigid or predefined release management process. The following sections define two example/possible release management process strategies. Either strategy can be used to produce good results while maintaining the ease of use of the overall system. The user of the system is free to use either one or a combination of the two strategies, depending on the needs of their organization.

Strict Release Management Process

A strict release management process is one that guarantees that any previous release can be directly extended to include new works or bug fixes. Recall that releases are built with change requests, and that change requests are made up of particular versions of files. Recall also that releases are based on previous releases, that is, release 2.0 is dependent on the particular versions of files in release 1.0 and so on. Once the 2.0 release has been formed (files have been extended from release 1.0 to form this new release), release 1.0 can be directly extended when a strict release management process is in place. **This strict release management process requires a new generic for each release.** This is also known as the **Branch on Release Pattern**[1]

Simple Release Management Process

A simple release management process is very similar to the strict process except that multiple releases can be formed in the same generic. When multiple releases are formed in a single generic, only the last release can be *directly* extended. Any one of the previous releases can be extended by forming a new generic with the files contained in that release. When a generic is formed from a release, the files in that generic will exactly match the versions of the files in the release.

When generics are created from releases in **SpectrumSCM™**, only the files that have actually changed from the baseline will get new disk images. All other files will simply be common object references to already existing files. This is done as a space saving feature considering that a good number of files in any system rarely change as new features are added.

Combinations of strict and simple release management

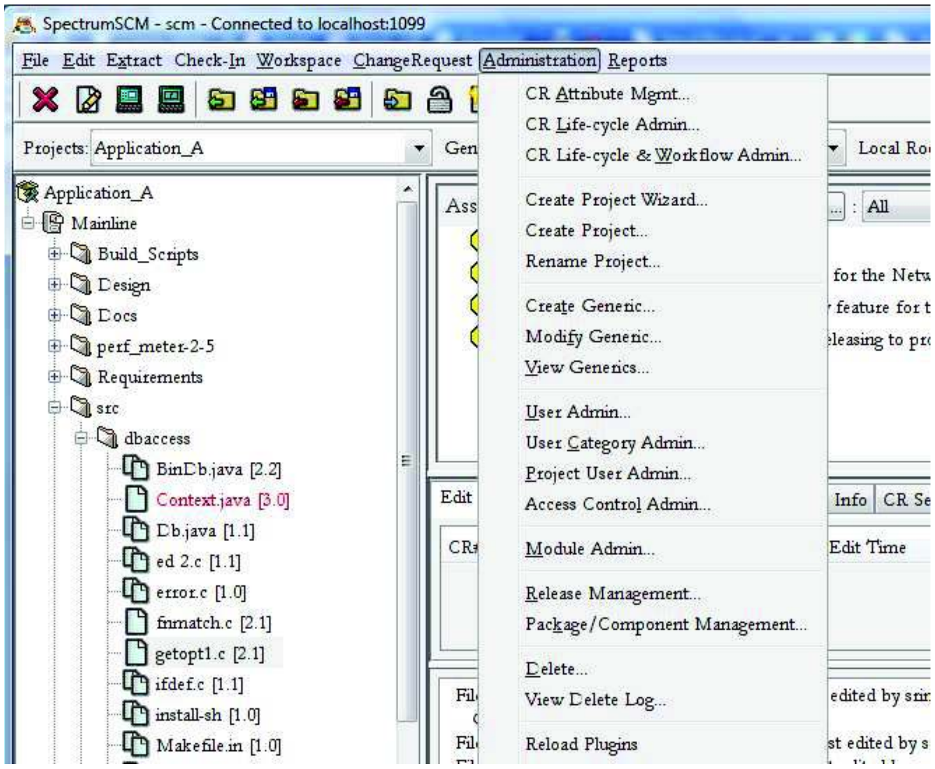
Combining the two release management process strategies involves using the simple strategy to produce patch releases of a product. When a new feature is being added, create a new generic (per the strict release management process) and do the new feature work in the second generic. The default behavior of the system is to check files out uncommon. This guarantees that the last major release of the product in any generic is always easily extensible in case the need arises. A fix applied to the current release (the last release of the previous generic) can be made common to the new generic by making the fix common.

[1] Berczuk, Stephen P. and Appleton, Brad. Software Configuration Management Patterns. Effective Teamwork, Practical Integration. Addison Wesley 2003. ISBN 0-201-74117-2

9.4 Creating a Release

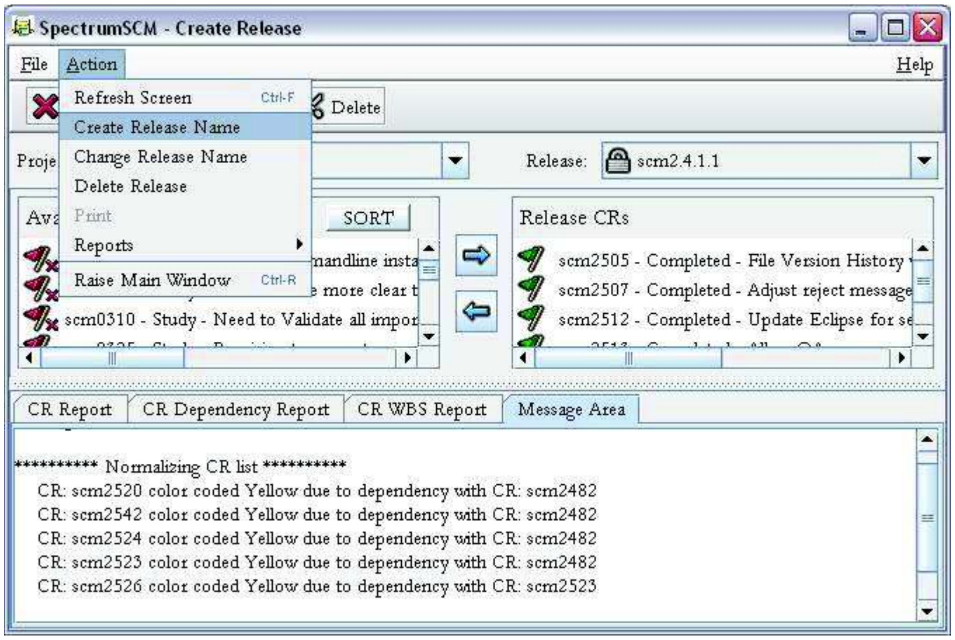
To create a release,

1) Select Release Management from the Administration menu on the Main Screen.



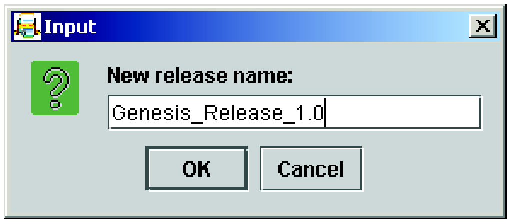
This will bring up the **Create Release Screen.**

2) Select **Create Release Name** via the Action Menu on the Create Release screen.

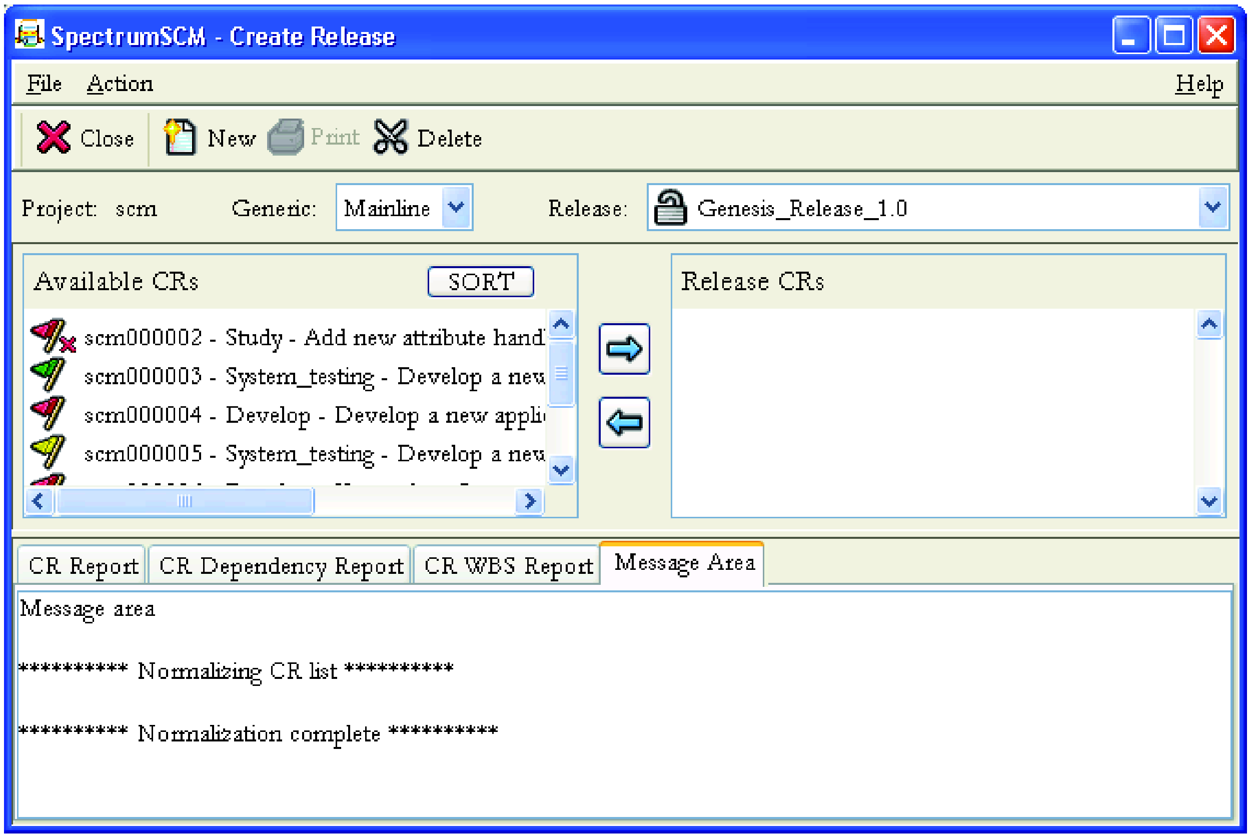


```
********** Normalizing CR list **********
  CR: scm2520 color coded Yellow due to dependency with CR: scm2482
  CR: scm2542 color coded Yellow due to dependency with CR: scm2482
  CR: scm2524 color coded Yellow due to dependency with CR: scm2482
  CR: scm2523 color coded Yellow due to dependency with CR: scm2482
  CR: scm2526 color coded Yellow due to dependency with CR: scm2523
```

3) This will bring up the **Input** window where you enter the new release name. Click **OK**.
This will take you back to the **Create Release screen.**
Notice that the release name is populated.



9.4.1 Select the CRs to be included in the release



Project: scm Generic: Mainline Release: Genesis_Release_1.0

scm000002 - Study - Add new attribute hand
scm000003 - System_testing - Develop a new
scm000004 - Develop - Develop a new appli
scm000005 - System_testing - Develop a new

Message area

********** Normalizing CR list **********

********** Normalization complete **********

Available CRs – This area displays the CRs that are in this generic. Note that CRs are flagged and the flags can be green, red, or yellow.

Green – the CR's development life cycle is complete (but possibly pending testing, approval and deployment type phases). You can include this CR and its associated files in the release.

Yellow - the CR has completed development; it is dependent on other CRs that have not yet been completed.

Red – the CR is not complete.

Any CR with a red "**x**" to the right of the flag icon indicates that the CR has no file level changes associated with it.

Choose the CRs that you want to include in the release and use the right arrow ⇨ to move them into the **Release CRs** area. If you make a mistake, you can use the left arrow to remove a CR from the release.

Only green CRs may be selected for inclusion in a release. If a CR that you wish to include in the release shows up as red ( ), it is not finished "development" i.e. it is not past the final source modification phase defined in the project's life-cycle. If this CR is wanted to be included in this release then the current assignee will need to finish their work and progress the CR or you (or someone else with assignment privileges) will need to use the Assign/Modify screen to move the CR past the final source modification phase. Requiring CR's to be past the final source modification phase guarantees the stability and integrity of the release build process because those CRs cannot be edited.

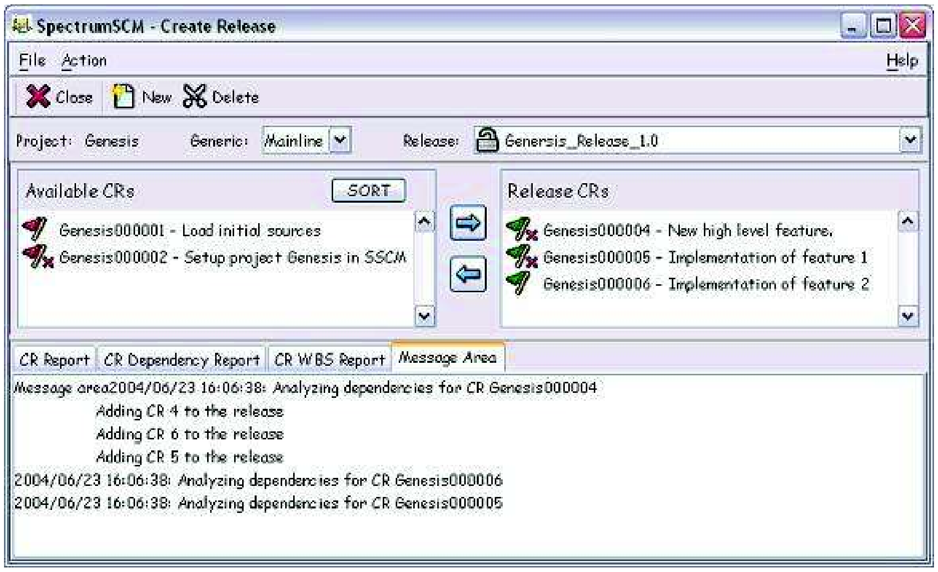
Note, it is perfectly valid for development to continue on CRs not needed by this release. These CRs can even edit files to be released in this release since they will be newer file versions and not dependent file versions.

If the CR is flagged in yellow ( ), it has one or more dependencies. Double-click the CR to show the dependency detail or right-click and run the specific **CR Report** or **Dependency** report. A yellow CR means that this CR has completed development, but it is dependant on other CRs that have not yet finished.

9.4.2 Dependency Checking

Dependencies are created when multiple CRs have made changes to the same file(s) or when CRs are involved in a work breakdown structure.

As you select CRs for the release, file level dependency checking is being performed. Only green (finished development) CRs and their dependants can be assigned. When a CR is placed into a release, any CRs that the original CR depends on will automatically be placed in the release, too. This action can be seen by opening up the message tab on the bottom of the release management screen.



Project: Genesis Generic: Mainline Release: Genersis_Release_1.0

Available CRs
- Genesis000001 - Load initial sources
- Genesis000002 - Setup project Genesis in SSCM

Release CRs
- Genesis000004 - New high level feature.
- Genesis000005 - Implementation of feature 1
- Genesis000006 - Implementation of feature 2

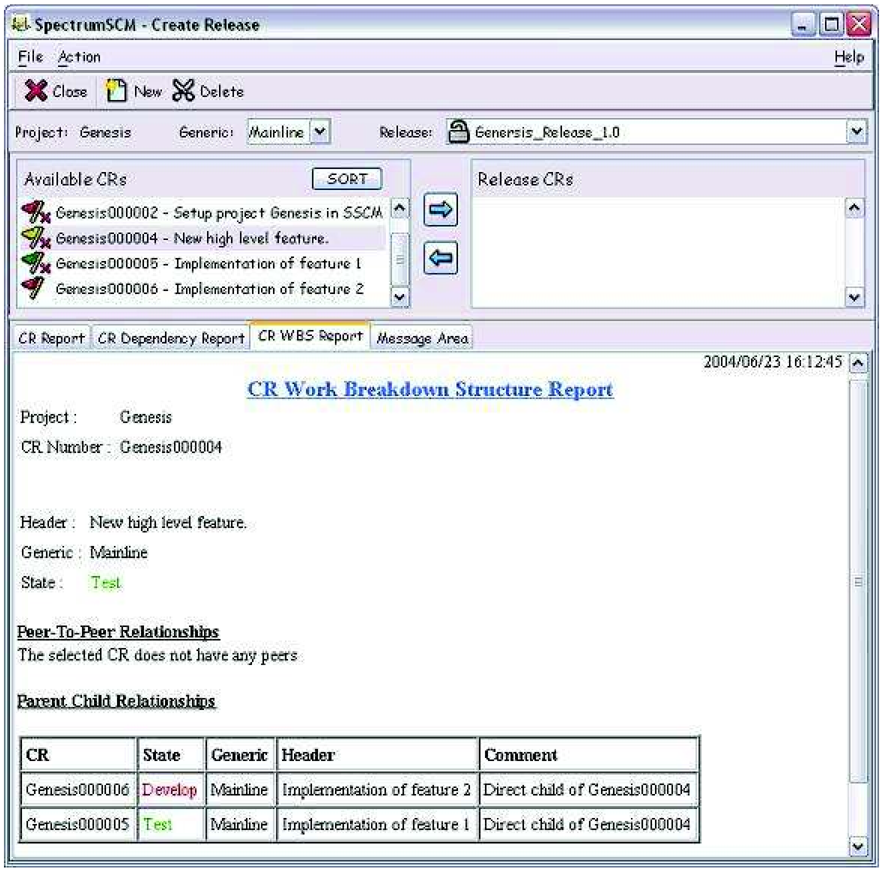
```
Message area2004/06/23 16:06:38: Analyzing dependencies for CR Genesis000004
            Adding CR 4 to the release
            Adding CR 6 to the release
            Adding CR 5 to the release
2004/06/23 16:06:38: Analyzing dependencies for CR Genesis000006
2004/06/23 16:06:38: Analyzing dependencies for CR Genesis000005
```

The above example shows that when CR Genesis000004 was added to the release that CRs 5 and 6 were automatically added to the release. This is the result of a parent/child relationship forming a dependency at the CR level. The same action will take place when CRs with file level dependencies are added to a release.

Requiring dependents to be added to the release guarantees the integrity of the build because nothing is being missed out. I.E. Feature A is dependent on Feature B but only Feature A gets placed into the release, this either breaks the build (or worse, the product at run time) because that dependency is unresolved – SpectrumSCM does not allow this situation to occur.

Once the set of incomplete CRs and dependant CRs have been identified, the developers responsible for completing these work items can be alerted and their progress monitored to ensure that they are going to complete the tasks on time. When the dependant CRs have been completed (past the last development phase), a refresh of the Release Management screen will show the CRs are now in a green state and they can be moved into the release.

In this next example, we are defining Genesis_Release_1.0. We see that CR Genesis000004 is yellow. When we right-click on Genesis000004 and look at the WBS Report, we see that it is involved in a parent/child relationship with CRs 5 and 6. CR Genesis000006 is still in the Develop phase. Therefore, CR Genesis000006 has to be completed before Genesis000004 can be moved into the release. CR Genesis000005 may be moved into the release since it is green flagged and has no dependencies.


SpectrumSCM - Create Release
File Action Help
Close New Delete
Project: Genesis Generic: Mainline Release: Genersis_Release_1.0
Available CRs SORT
Release CRs
CR Report CR Dependency Report CR WBS Report Message Area

Genesis000002 - Setup project Genesis in SSCM
Genesis000004 - New high level feature.
Genesis000005 - Implementation of feature 1
Genesis000006 - Implementation of feature 2

2004/06/23 16:12:45

CR Work Breakdown Structure Report

Project : Genesis

CR Number : Genesis000004

Header : New high level feature.

Generic : Mainline

State : Test

Peer-To-Peer Relationships

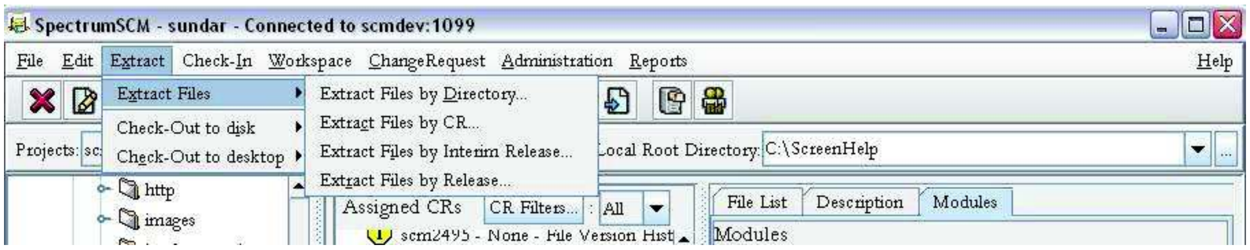
The selected CR does not have any peers

Parent Child Relationships

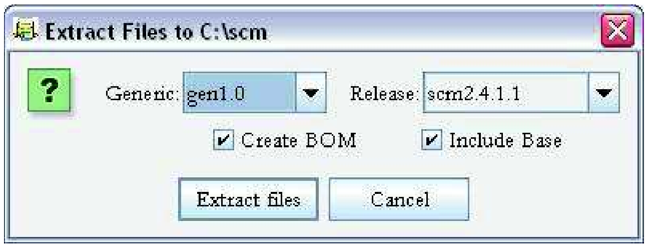
CR	State	Generic	Header	Comment
Genesis000006	Develop	Mainline	Implementation of feature 2	Direct child of Genesis000004
Genesis000005	Test	Mainline	Implementation of feature 1	Direct child of Genesis000004

9.4.3 Extract Files to Build the Release

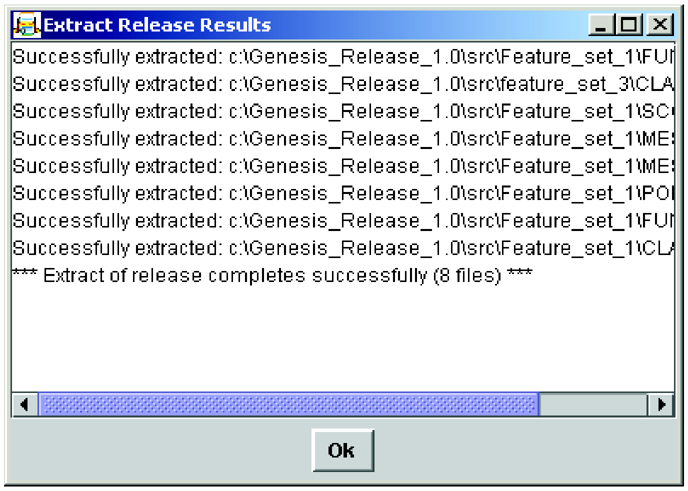
Once a release has been defined, an **Extract Files by Release** can be performed (using the **Main Screen Extract / Extract Files by Release** menu option or cmmand-line routine **scm_gr**) to extract the necessary files that can be used to build the release. The file versions associated with the CRs included in the release will be extracted to the Local Root Directory.


SpectrumSCM - sundar - Connected to scmdev:1099
File Edit Extract Check-In Workspace ChangeRequest Administration Reports Help
Extract Files
Check-Out to disk
Check-Out to desktop
Extract Files by Directory...
Extract Files by CR...
Extract Files by Interim Release...
Extract Files by Release...
Projects:
Local Root Directory: C:\ScreenHelp
http
images
Assigned CRs
CR Filters...
All
scm2495 - None - File Version Hist
File List
Description
Modules
Modules

NOTE: Do not extract files for building a release into the same local root directory used for development since this would potentially overwrite (or at least warn about) any on-going development files. Set up a different directory (remember, you can have multiple local root directories). The **Extract Files** window allows you to verify or choose the generic and release to extract.



Click **Extract Files**. All files associated with the CRs included in the release will be extracted to the Local Root Directory specified. If the **Create BOM** and **Include Base** options are selected, a Bill of Materials for the release along with the files in the Base will be created under the local root directory. The BOM file uses a *SSCM_REL_BOM_Date_Timestamp* format for the file name. The results of the extract process are displayed.



```
Successfully extracted: c:\Genesis_Release_1.0\src\Feature_set_1\FU
Successfully extracted: c:\Genesis_Release_1.0\src\feature_set_3\CLA
Successfully extracted: c:\Genesis_Release_1.0\src\Feature_set_1\SC
Successfully extracted: c:\Genesis_Release_1.0\src\Feature_set_1\ME
Successfully extracted: c:\Genesis_Release_1.0\src\Feature_set_1\ME
Successfully extracted: c:\Genesis_Release_1.0\src\Feature_set_1\PO
Successfully extracted: c:\Genesis_Release_1.0\src\Feature_set_1\FU
Successfully extracted: c:\Genesis_Release_1.0\src\Feature_set_1\CL
*** Extract of release completes successfully (8 files) ***
```

9.5 Building the Release

SpectrumSCM gives users the flexibility to use their native environment and desired products to do their builds. Extract the files associated with a release to the local directory from which you would normally run your build. After extracting the files, use them as input into the desired build process to compile the release.

9.6 Adding CRs to a Release

Once a release has been created, CRs can be added via the **Create Release** screen. New files can be added to existing CRs. New CRs can be created and added to the release.

To add, a new feature or bug fix to an existing release, simply add the CR for that feature or bug fix to the **Release CR** list. When CRs or files are added, changed or removed from a release, you must extract the code and rebuild the release for the new code to be included.

NOTE: CRs cannot be added to or removed from a release that has been locked. Release 1.0 will be locked when Release 1.1 is created. Creating a new generic locks the last release in the previous generic.

9.7 Removing CRs from a Release

CRs can be removed from a release via **the Create Release** screen. Use the left arrow to remove a CR from the release. When CRs or files are added, changed or removed from a release, you must extract the code and rebuild the release for the new code to be included.

NOTE: CRs cannot be added to or removed from a release that has been locked. Release 1.0 will be locked when Release 1.1 is created. Creating a new generic locks the last release in the previous generic.

9.8 Re-creating a Past Release

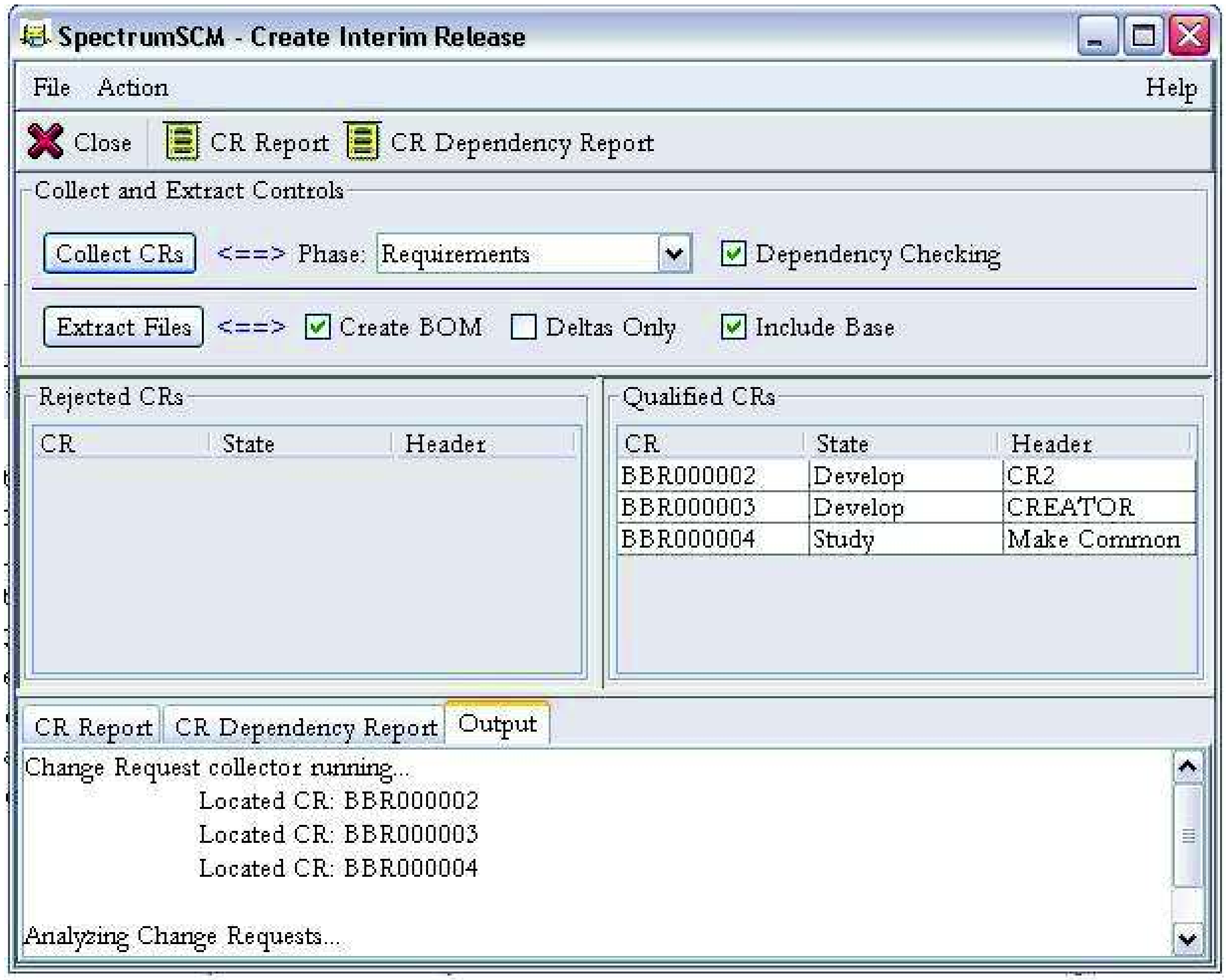
Any release defined in SpectrumSCM can be re-created by following the same steps used for extracting files in a new release. Use the **Main Screen Extract / Extract Files by Release** menu option to pull up the **Create Release** screen. Set up a local root directory specifically for the release so that files belonging to other releases are not corrupted (remember, you can have multiple local root directories). Choose the generic and release to extract using the pull down boxes for generic and release and click **Extract Files**. All files associated with the CRs included in the selected release will be extracted to the specified Local Root Directory.

The results of the extract process are displayed. The contents of the specified local root directory can then be used as input into the build process to recreate the release.

9.9 Interim Releases

You use Interim Releases to extract a set of files based on the phase of your life-cycle but before they are placed into a formal release. So for example, if you have an "Integration Test" life-cycle phase as part of your process, but developers can still modify files under these CRs until things stabilize and become ready for the formal release.

Since this is an informal process, you also have various options to override the dependency checking if needed. Specifically you can turn-off dependency checking altogether (via the checkbox at the top of the screen) or via a right mouse click context menu, include or exclude specific CRs as necessary. Note that by including CR's with dependency issues you might get undesirable end-results. Changes from the dependant edits could affect compilation or execution of the end product. Select the phase in your life-cycle for which you wish to create this IR and then press the "**Collect CRs**" button. The qualified and rejected CRs will be displayed in the right and left panels respectively.



Phase: Requirements

[✓] Dependency Checking

[✓] Create BOM [ ] Deltas Only [✓] Include Base

Rejected CRs

CR	State	Header

Qualified CRs

CR	State	Header
BBR000002	Develop	CR2
BBR000003	Develop	CREATOR
BBR000004	Study	Make Common

```
Change Request collector running...
                    Located CR: BBR000002
                    Located CR: BBR000003
                    Located CR: BBR000004

Analyzing Change Requests...
```

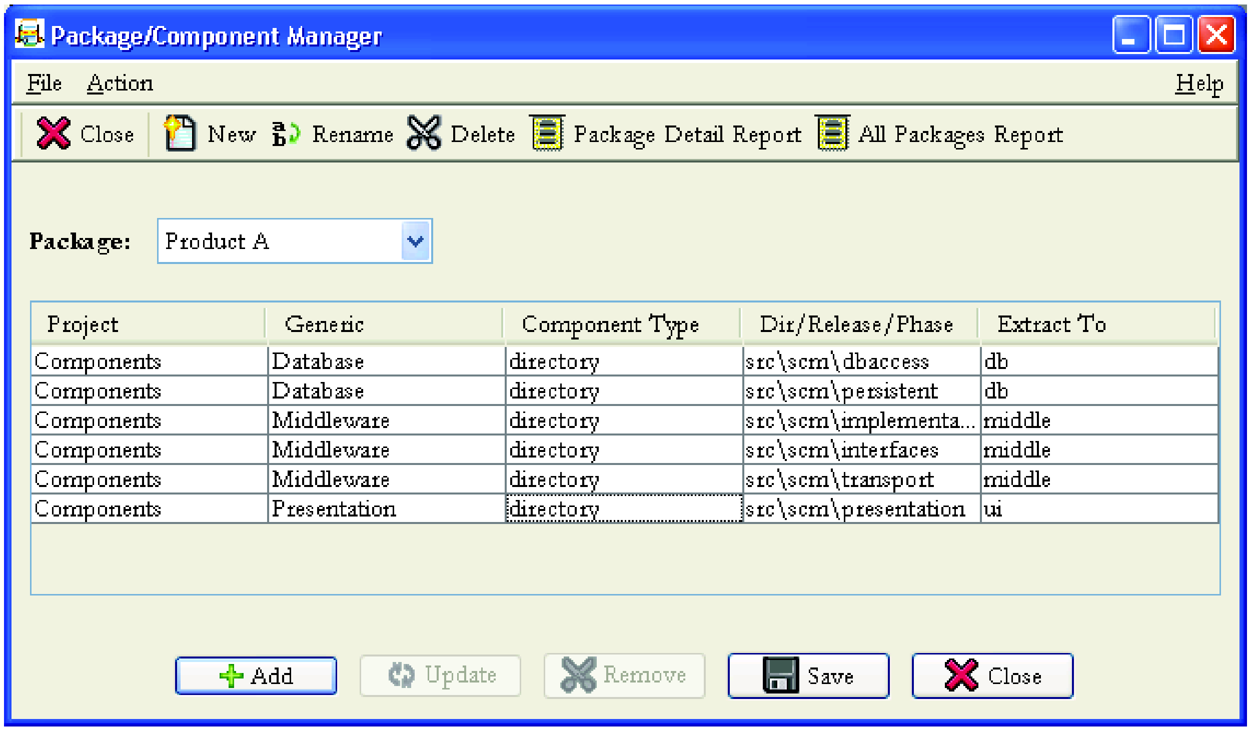
The "**Create BOM**" checkbox will create a Bill of Materials report as part of the extraction. This lists all the files and their version numbers that make up this extraction. Additionally, by selecting the "**Deltas Only**" checkbox, if you are performing a series of extractions and builds, only those files that have changed since the previous extraction will now be pulled. This could be a significant performance improvement, depending on your process.

The interim release command line command can also be used to automate nightly-build or continuous development environments. Either nightly or as soon as a CR is progressed from the development phase the **scm_gir** command could be triggered to extract and build the target environment.

9.10 Package/Component Management

The Package/Component management feature allows the product manager to organize the SpectrumSCM controlled assets in a flexible manner within the overall repository. For example if your product consists of 3 separate components (it could consist of many more), a database component, a web servlet/ASP component and a main heavy-weight user interface component. These 3 components could be controlled within SpectrumSCM as separate folders, generics or even projects. Depending on the dependencies within these components or their independence, how to store them might vary from one organization to another.

Release Management (as described above) allows the Generic Engineer/Release Manager to define release sets based on Change Requests within a particular project and generic. If you want to organize your components above this level then package management can be used to bring the overall product together in an appropriate manner.



Package: Product A

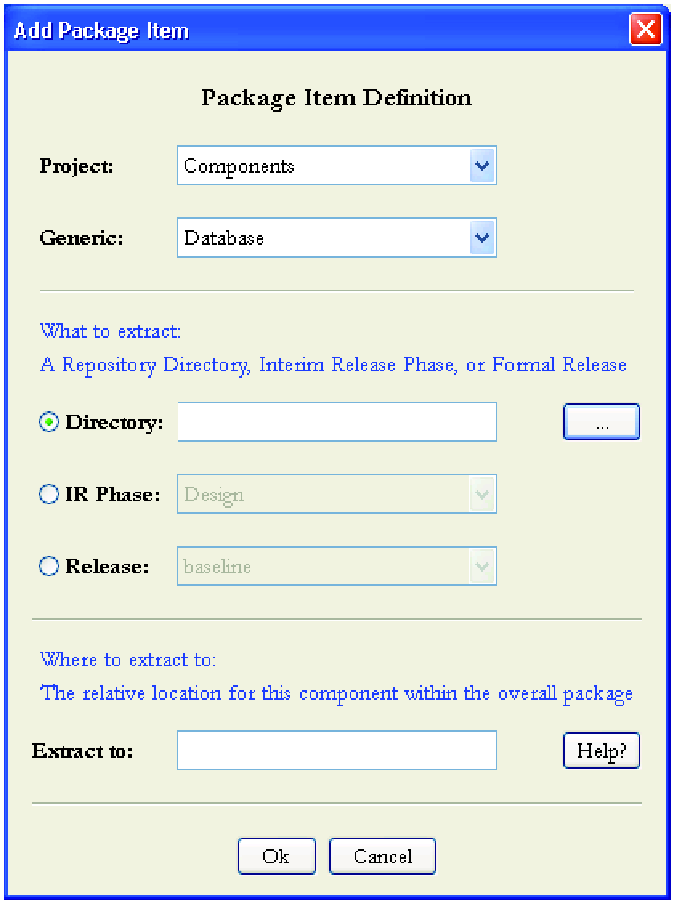
Project	Generic	Component Type	Dir/Release/Phase	Extract To
Components	Database	directory	src\scm\dbaccess	db
Components	Database	directory	src\scm\persistent	db
Components	Middleware	directory	src\scm\implementa...	middle
Components	Middleware	directory	src\scm\interfaces	middle
Components	Middleware	directory	src\scm\transport	middle
Components	Presentation	directory	src\scm\presentation	ui

Once a package has been defined it can be extracted in a reproducible manner via the "Extract -> Extract Files -> Extract Files by Package" menu option or the **scm_gpack** command-line.

A Package can be defined by hitting the "New" button or the "File -> Create Package" menu item. This defines an empty package to which you can "Add" the appropriate components (or Action menu -> Add Package Item). This will present the Package Definition window as shown below.

Existing packages can be updated or even removed, however all such actions will be logged (in the server logs directory) for auditability purposes.

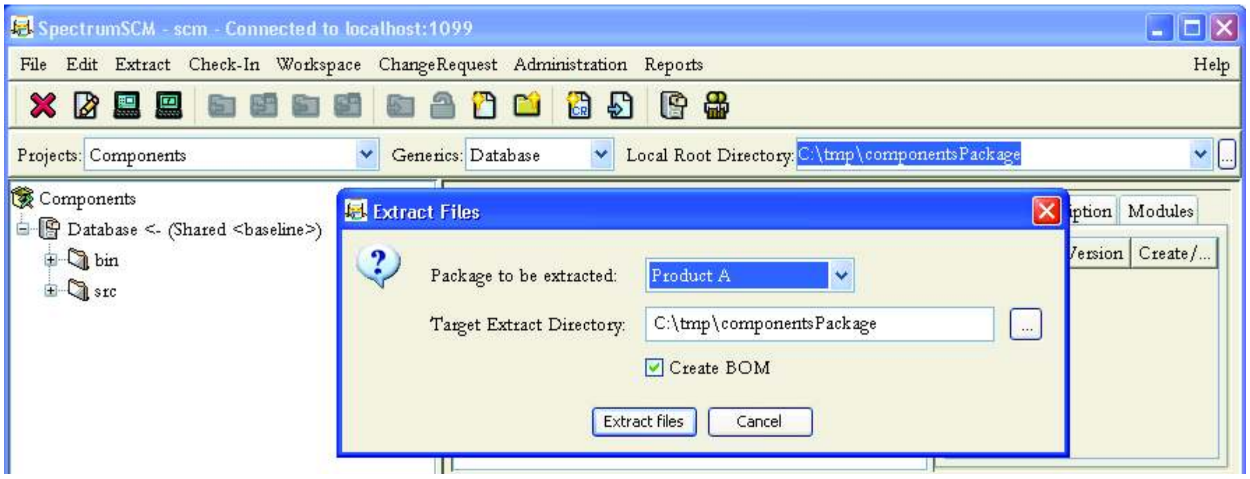
A package item or component can be any release, directory or even an interim release (for testing builds). Select the appropriate project and generic, and then the appropriate radio button (on the left-hand side) to indicate which type of component this package item is to be selected from.



For a release or interim-release, the selection is a simple choice-box. Select the appropriate release or IR phase in these cases. For a directory component, the directory path can either be typed in, or the "…" button can be used to browse for the desired repository folder(s). If more than one directory folder is selected, each will be represented as their own component in the overall package definition (for example the 2 database folders or 3 middleware folders shown above).

Once a package is defined it can be extracted through the Extract menu. In performing a package extract a "**target extract directory**" is specified. This defaults to the current local root directory but can be changed. When performing an extract each component is extracted to the target extract directory **PLUS** the "**Extract to**" location (if any is specified). Thus if the target extract directory is specified as below, then the database components in the example above will be extracted to the `C:\tmp\componentsPackage\db` folder.

The "**Extract to**" location for a particular package component is the **relative** directory into which it should be extracted.


SpectrumSCM - scm - Connected to localhost:1099
File Edit Extract Check-In Workspace ChangeRequest Administration Reports Help
Components
Database <- (Shared <baseline>)
bin
src
Modules
Version
Create/...

Projects: Components
Generics: Database
Local Root Directory: C:\tmp\componentsPackage

Extract Files

Package to be extracted: Product A
Target Extract Directory: C:\tmp\componentsPackage
☑ Create BOM

When extracting a package a Bill of Materials report will be produced (unless the "Create BOM" option is un-checked) just like the other extraction methods (release and/or IR). The BOM includes a textual report (file list) with the name "SSCM_PKG_FL_*date_time*.txt" and the full HTML report with the name "SSCM_PKG_BOM_*date_time*.html". The BOM reports detail all the files extracted and their specific version numbers and locations.

The BOM is also available on-demand directly from the package management screen "**Package Detail Report**" button. The "**All Packages Report**" produces a summary report of all the packages and their components as they are currently defined.

Package Management is aligned with Release Management and the Generic Engineer role from the perspective of permissions. Therefore only users with "Create New Generic" permissions will be able to access the Package Management screen. The user will then only be able to access the projects to which they have the "Create New Generic" permissions in that project. Administrators and system-level Project Engineers have this permissions level for all projects.