

A guide for sharing components in Plastic SCM.

Plastic SCM 4

Plastic SCM **Xlinks guide**

© 2006-2012 Codice Software

Warning and Disclaimer

Every effort has been made to make this book as complete and as accurate as possible, but no warranty or fitness is implied. The information provided is on an “as is” basis. The author and the publisher shall have neither liability nor responsibility to any person or entity with respect to any loss or damages arising from the information contained in this book.

Trademarks

All terms mentioned in this book that are known to be trademarks or service marks have been appropriately capitalized. Codice Software cannot attest to the accuracy of this information. Use of a term in this book should not be regarded as affecting the validity of any trademark or service mark.

Table of Contents

[**Chapter 1** What is an Xlink 2](#_Toc317867998)

[**Chapter 2** Creating Xlinks 4](#_Toc317867999)

[2.1 Modifying an Xlink 6](#_Toc317868000)

[2.2 Exploring the Xlinked repository 6](#_Toc317868001)

[**Chapter 3** Types of Xlinks 8](#_Toc317868002)

[3.1 Writable Xlinks 8](#_Toc317868003)

[3.1.1 Branch auto-expansion 9](#_Toc317868004)

[3.1.2 Branch auto-expansion naming rules 11](#_Toc317868005)

[3.1.3 Creating a writable Xlink 13](#_Toc317868006)

[3.1.4 Merging 13](#_Toc317868007)

[**Chapter 4** Relative Xlinks 17](#_Toc317868008)

List of Figures

[Figure 1: sample Xlink 2](#_Toc317867982)

[Figure 2: Several Xlinked repositories 3](#_Toc317867983)

[Figure 3: Xlink evolution 3](#_Toc317867984)

[Figure 4: Xlink definition syntax sample 4](#_Toc317867985)

[Figure 5: Just created Xlink in the Pending Changes view 5](#_Toc317867986)

[Figure 6: Xlink in the Items view 6](#_Toc317867987)

[Figure 7: Repository menu for an item in an Xlinked repository 7](#_Toc317867988)

[Figure 8: Writable Xlink sample scenario 9](#_Toc317867989)

[Figure 9: Change checked in with a Writable Xlink 10](#_Toc317867990)

[Figure 10: Diff changeset contents shows changes in Xlinked repositories 10](#_Toc317867991)

[Figure 11: Branch auto-expansion in repositories with different branch structure 12](#_Toc317867992)

[Figure 12: Read-only vs. writable-Xlinks in the Items view 13](#_Toc317867993)

[Figure 13: Sample conflict: same file modified on two branches on Xlinked repositories 14](#_Toc317867994)

[Figure 14: branch "task0127" is merged to "task0243" 15](#_Toc317867995)

[Figure 15: Merging changes in Xlinked repositories 15](#_Toc317867996)

[Figure 16: Merged branches in top-level and WXlinked repositories 16](#_Toc317867997)

# What is an Xlink

Projects often need to reuse existing components that have been developed and are actively used as part of other projects. Plastic SCM encourages using separate repositories for different projects and components, and access shared components by “mounting” them in the project repository through Xlinks. This chapter will introduce what are Xlinks and how to use this powerful feature to share components among projects that evolve in parallel.

An Xlink is pretty similar to a symbolic link, like those found in Unix operating systems. It is a directory entry in your repository tree that points to another directory in a different repository. An Xlink, however, will also contain information about the specific **version** of the target directory to which it is pointing.



Figure : sample Xlink

Xlinks can point to repositories with other Xlinks inside them, so Xlink component mounting can address really complex development scenarios.



Figure : Several Xlinked repositories

Xlinks pointing to a given version of the target directory let the user specify, for instance, that project X on version 1.1 is using version 2.1 of library Y. When a new version of the library is labeled, say 2.2, the code of the project can be updated to use it and maybe version 1.2 of project X uses library Y 2.2.

Since Xlinks are versioned, the Xlink in old project X version 1.1 still points to library Y 2.1 and the original configuration can be completely rebuilt without any issue. The following figure depicts how it looks like:



Figure : Xlink evolution

# Creating Xlinks

An Xlink is defined using the Plastic SCM command line client, through the ‘**xlink’** command. An Xlink is defined with at least two arguments:

1. The directory entry to create in the current repository. This is a directory that will be used to
2. The directory to mount in the target repository. In the current implementation this is limited to the root directory, but this situation will most likely change in future releases of Plastic SCM to support any directory in the target repository.
3. Version and repository to mount at point 1 above. The version of the root directory to mount is specified using either a changeset specification or a label specification.



Figure : Xlink definition syntax sample

The command used to create the Xlink in the sample referred in section looks like this:

cm xlink component1 / 1@mylibrary

In the sample:

* component1 is the directory in the project repository that will point to the mounted mylibrary component. This directory must not exist already in the repository. The “xlink” command will create an item for it and it will throw an error if a file of directory already existed with the same name.
* / is the directory in the target repository, as described above.
* 1 is the changeset number in the target repository.
* mylibrary is the target repository. Together with the changeset it forms a changeset specification. In this case, there is no server specified. If the server is to be specified, the spec would look like this:
  + 1@mylibrary@**servername:8087**
  + It is also possible to specify a label instead of a changeset. Since a label is actually a pointer to a changeset, the two specifications are interchangeable. To do so
  + Keep in mind that specifying an xlink with a label only uses the label to retrieve the changeset to which it points and sets the xlink to it. This means that if the label is moved afterwards, the Xlink will still point to the original changeset.

The xlink is created in the workspace and appears as a pending change in the pending changes view or to the cm status –a command, as depicted in the following figure:

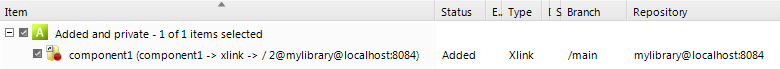


Figure : Just created Xlink in the Pending Changes view

Once it is checked in, the Xlink needs to be updated so the contents of the mounted directory are downloaded into the workspace. This can be done in the GUI by right clicking on the Xlink and selecting “Update”. A complete workspace update or an update of any parent directory will also populate the contents of the Xlink.



Figure : Xlink in the Items view

## Modifying an Xlink

The target of an existing Xlink can be modified using the “-e” modifier of the xlink command. The syntax in this case is similar to the creation syntax, but the Xlink must exist already. The following example modifies an existing Xlink “component1” to point to changeset 5 on repository “mylibrary”.

cm xlink –e component1 / 5@mylibrary

## Exploring the Xlinked repository

The Plastic SCM GUI views that you can open in the left options tree display the information of the top-level repository loaded in the workspace. But it may be desirable to see the Branch Explorer or the Changesets views of an Xlinked repository.

The items view makes this easy through the “Repository” submenu. When you right click on an item inside an Xlinked repository, the “Repository” submenu lets you open the most important views for the repository where the item is located: i.e. the Xlinked repository. For your convenience, the first entry in the menu contains the repository name of the item.



Figure : Repository menu for an item in an Xlinked repository

# Types of Xlinks

If you try to make a change in any item of the mounted Xlink, you’ll get an error stating “It is not possible to make changes on a read-only xlink”. This is the case because the example used a read-only Xlink.

You can create two different types of Xlinks, depending on what you want to do with the target repository:

* **Read-only Xlinks** are recommended for situations where you are the user of an existing component, but you don’t influence its development. In the sample above, you were *Xlinking* an existing library and only calling it from your project, but *not making any changes to it*.
* **Writable Xlinks**, on the other hand, let you make changes inside the Xlinked repository. They are described in the next section.

## Writable Xlinks

Writable Xlinks (or WXlinks) let you perform changes in the Xlinked repository. But, where do these changes fit in the new repository? Imagine a simple example in which your top level repository is again “ProjectX” and your Xlinked repository is called MyLibrary. The “ProjectX” branch you are using in your workspace is “/main/task0127”, but this branch doesn’t exist in “MyLibrary”. If you try to make a change in that repository, where does the change go?



Figure : Writable Xlink sample scenario

### Branch auto-expansion

Xlinks solve this problem with a feature called Branch Auto-expansion. Now you add a new file in “Changes\_on\_127.txt” in your workspace, in the /component1 directory, or make any other change to existing files. Several things happen when you checkin the change in this scenario. Figure 9 summarizes the result.

1. Plastic SCM will automatically create a new branch in the Xlinked repository with the same name as the branch in the parent repository (“task0127”). The branch base is set to the changeset where the Xlink was pointing (changeset number 1 in repository “MyLibrary”). This is the “branch auto-expansion”.
2. In “MyLibrary” repository, a new changeset is created in the new branch with the added file. (changeset number 2 in “MyLibrary”, inside “task0127” branch).
3. In “ProjectX” repository, the Xlink is modified to point to the new changeset in the xlinked repository. As a result of this, a new changeset is created as well in repository “ProjectX” (changeset number 3 in ProjectX repo).



Figure : Change checked in with a Writable Xlink

So, indeed, two different changesets have been created, one in “ProjectX” and another one in “MyLibrary”. If you double click changeset 3 in the Branch Explorer to compare its changes, you’ll see that the Plastic SCM GUI displays not only the change in “ProjectX” but also the changes in the Xlinked repository. Figure 10 shows this.

Note that there are two changesets in this scenario, each on its own repository, but the “Compare changeset content” is displaying them together for your convenience.



Figure : Diff changeset contents shows changes in Xlinked repositories

**Branch auto-expansion is designed so that you can do all your operations on the workspace and they are auto-expanded to any Writable-Xlinks you have in your repository. Later sections will explain how merge, for instance, is auto-expanded as well to the Writable-Xlinked repositories. The overall goal of this design is that the user could work with mounted repositories performing all the operations on the top-level one. It is easier to operate in this repository alone and have the operations propagated to the Xlinked repositories.**

**It is important to note also that auto-expanded branches in the Xlinked repository are not really tied to the parent repository, so the Xlinked repository can be freely used as an independent repository as well.**

### Branch auto-expansion naming rules

What happens if the branching structure is different in the top level repository and the Xlinked repository? This situation is depicted in Figure 10. In this case:

* “ProjectX” contains an Xlink in branch /main/task0127 pointing to changeset number 2 in branch “/main/development” in “MyLibrary”.
* After checkin, branch auto-expansion creates a new branch in “MyLibrary”. Since the target changeset of the WXlink was in branch “/main/development”, the auto-expanded branch is created as a child of it: “/main/development/task0127”.



Figure : Branch auto-expansion in repositories with different branch structure

In general, the following rules apply when creating an auto-expanded branch:

* First, a check is made to see if a branch with the same full name exists in the target repository:
  + If it exists, this is used
  + If it does not exist, Plastic SCM builds the branch name this way:
    - name of the branch of the target xlinked changeset + short name of the top-level repository branch (last part).
    - If this branch exists, it is used as the auto-expanded branch.
    - Otherwise, the branch is created and the branch base is set to the Xlinked changeset.

### Creating a writable Xlink

Writable Xlinks are created in the command line, as read-only ones, but you have to specify the “-w” modifier:

cm xlink –w component1 / 1@mylibrary

You’ll notice in the Items view of the GUI client that the Type column contains “wxlink” where it used to be “xlink” for read-only Xlinks.



Figure : Read-only vs. writable-Xlinks in the Items view

### Merging

Merging a branch also affects the auto-expanded branches in the WXlinked repositories. The merge operation is done on the top-level repository (the repository to which the workspace is pointing to). It will then evaluate any new versions of WXlinks that have been created and consider the changes in the linked repositories in the merge process as well.

In this sample scenario, we have the same 2 repositories we used in previous sections: “ProjectX” and “MyLibrary”. “ProjectX” is the top-level repository and contains a WXlink “component1” pointing to “MyLibrary”. In “ProjectX” we have created two child branches: task0127 and task0243.

We then made changes in the file “Form1.cs”, located in the Xlinked “MyLibrary” repository in both branches. This, indeed, expanded branches task0127 and task0243 to the “MyLibrary” repository.



Figure : Sample conflict: same file modified on two branches on Xlinked repositories

Now, what happens when you try to merge from branch task0127 to branch task0243, as depicted in Figure 14?



Figure : branch "task0127" is merged to "task0243"

The merge operation will consider the changes made to the WXlink in both branches and then evaluate the changes in the WXlinked repositories themselves to include them in the merge. So, as a user, you’ll have to resolve any possible conflicts as you would with a normal “single repository” merge.



Figure : Merging changes in Xlinked repositories

Now the merging of Form1.cs is resolved but, what about the WXlink? The answer is that the merge operation will also update the WXlink to point to the result of the merge in the MyLibrary repository. That is, as a result of the merge operation, there is a new version of Form1.cs with the merged changes, and the WXlink in the “ProjectX” repository will point to this new changeset. For this reason, the Pending Changes view shows a pending merge link for the “ProjectX” repository, as depicted in Figure 15.

Once the changes are checked in, two changesets are created:

* In the “MyLibrary” repository, a changeset is created with the result of the merged Form1.cs file.
* In the “ProjectX” repository, a changeset is created with the modified WXlink pointing to the result of the merge.



Figure : Merged branches in top-level and WXlinked repositories

The overall result is that you only need to tell Plastic SCM “I want to merge this branch” in your workspace, and the merge operation will handle the changes made in WXlinked repositories.

# Relative Xlinks

When Xlinks are replicated, the target they point to is still the original one. So, say that you are replicating a repository with an xlink “component1” pointing to changeset number 6 on repository “mylibrary” on server “mainserver.location1.com:8087”. The destination of your replica is on another Plastic SCM server at “otherserver.location2.com:8087”. When you replicate the top-level repository, the Xlink in your replicated repository will still point to “mainserver”. If it is reachable from the “location2.com” network, the data for “mylibrary” repository will be downloaded from “location1.com”.

Most of the time, you’ll want to replicate the “mylibrary” repository to “location2.com” as well, and perform any changes in the local replica.

To tell the Xlink that you want to use the “mylibrary” repository found in your default server (that of the top-level repository), you need to create the Xlink with a relative server.

This is achieved with the “-rs” modifier when the Xlink is created or edited, as in the following example:

cm xlink –rs component1 / 6@mylibrary

When this link is replicated, it’ll try to locate “mylibrary” repository in the local server, rather than “mainserver.location1.com”.

Note that changeset number 6 may no longer have number 6 in the replicated repository (changesets are internally identified by their GUID, not their number, since replicas may have different numbering. Figure 10 in the previous section already depicted this. Refer to the Plastic SCM Distributed System Guide for more details). The Xlink will be pointing to the right changeset even if the number changes, because it uses the changeset GUID rather than the changeset number to locate it.