



CM
bridge
User Guide

Contents

1 Introduction	1/12
1.1 Overview	1/12
1.2 ClearCase to Subversion/Git	1/12
1.3 Subversion to Subversion	3/12
2 Merge Conflicts	4/12
3 Problem Types and Resolutions	5/12
3.1 Subversion File Merge Conflict	5/12
3.1.1 Signature	5/12
3.1.2 Resolution	6/12
3.2 Subversion Property Merge Conflict	6/12
3.2.1 Signature	6/12
3.2.2 Resolution	7/12
3.3 Subversion Lock Warning	7/12
3.3.1 Signature	8/12
3.3.2 Resolution	8/12
3.4 ClearCase Merge Conflict	8/12
3.4.1 Signature	8/12
3.4.2 Resolution	8/12
3.5 Other ClearCase Errors	8/12
3.5.1 Signature	8/12
3.5.2 Resolution	9/12
3.6 Other Subversion Errors	9/12
3.6.1 Signature	9/12
3.6.2 Resolution	10/12
3.7 Git Merge Conflict	10/12
3.7.1 Signature	10/12
3.7.2 Resolution	11/12
3.8 Git Ahead Errors	11/12
3.8.1 Signature	11/12
3.8.2 Resolution	11/12
4 Background Information	12/12

1 Introduction

Welcome to Clearvision's CM Bridge products: Integrating Subversion, Git and ClearCase.

The CM Bridge provides Clearcase to Subversion (CC2SVN), Clearcase to Git (CC2GIT) and Subversion to Subversion (SVN2SVN) Bridging.

This user guide gives information for *users* of repositories being bridged, and in particular important information that is necessary for dealing with "Merge Conflicts" within bridges. The latest version of this document is available [on our website](#)

For details on bridge setup, planning, and installation, please refer to The CM Bridge [Administration Guide](#).

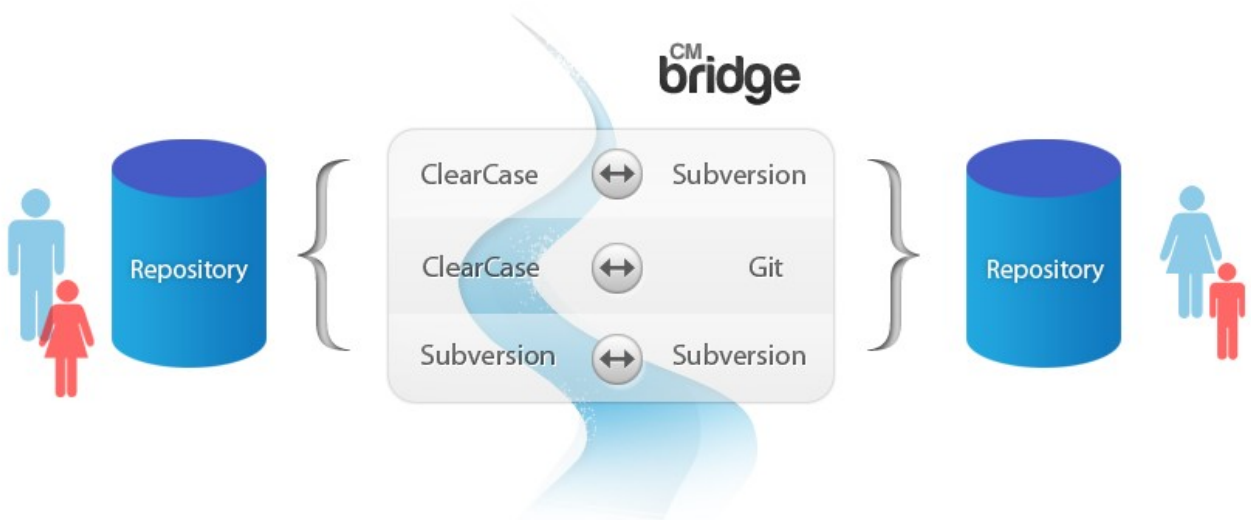
1.1 Overview

The CM Bridge transparently synchronizes data between ClearCase, Subversion and Git. Hence, you can have one part of a team working with Clearcase, and another part working with Subversion or Git, and they can *all* work together seamlessly, without knowing that some people are using another type of SCM system.

1.2 ClearCase to Subversion/Git

So, for example, as a normal user of Subversion, you carry on making changes using your normal Subversion tools. The only difference you will see is that members of the team working at the ClearCase end of your bridge may also make changes, and these will appear in your Subversion repository simply as if another user has made those changes.

The CM Bridge creates links between a ClearCase view on one side and a Subversion or Git workspace on the other as shown in the following diagram.

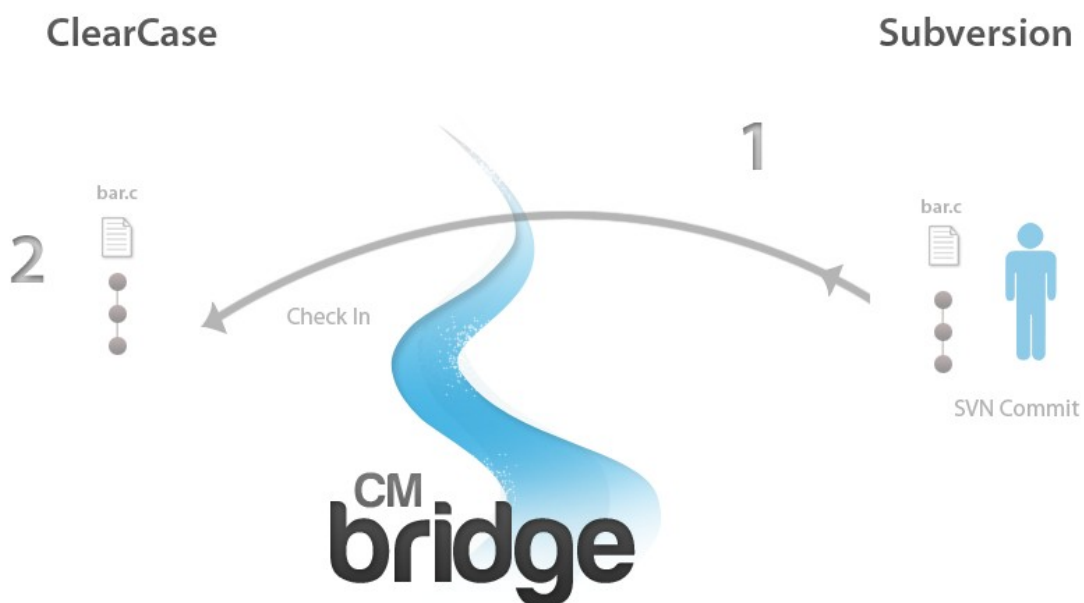


As users work normally within their ClearCase, Subversion and Git environments, data is transferred automatically in the background. This process should be fully transparent and users do not need to do anything special for bridging to take place.

When data is checked in to one side of the bridge (in this case ClearCase), a regular update process will pick up these changes and transfer them to the other side (Git or Subversion).



Similarly, when changes are committed to the 'other' side of the bridge (Git or Subversion), the bridge will pick up the new data on the next regular update and transfer it back.



1.3 Subversion to Subversion

CM Bridge also provides functionality for Subversion to Subversion bridges; therefore you can effectively bridge data between subversion repositories operating on different servers/sites that may be located on different continents. This functionality is similar to the Clearcase to Subversion functionality shown above, however both sides of the bridge are Subversion (as shown by the following Diagram)



This gives you the ability to provide the functionality of a Subversion multi-site without having to install additional applications.

2 Merge Conflicts

Most of the time, the changes from either side of a bridge will not overlap, so the bridge will work transparently. However, if the same data is being modified at both ends of a bridge, *and these changes conflict*, users will need to intervene to resolve these conflicts. (This is the same situation as 2 people making conflicting changes while working on the same files in one repository). Each Bridge that is created can be configured with a list of email addresses ("resolvers") for this purpose.

In this scenario, the set of "resolvers" (set up by the Administrator) for a bridge are informed via email, and it is their responsibility to communicate with the teams at either side to resolve the conflict by applying a "Fix Activity" (ie a change that fixes the conflict). You might set a project lead as a sole resolver, or you might have a distribution list that goes to all team leads on a project, or you can even have all of your individual developers on the resolvers list if you like. However you prefer to set this up, it is important that the people being emailed can arrange for the problem to be fixed as soon as a problem arises.

The next section explains how to deal with the most common conflicts.

3 Problem Types and Resolutions

As data is being transferred in the background by a process not normally observed by users, email notifications are provided should a bridge encounter any problems.

Email notifications contain a list of all the issues encountered during a bridge update.

Potential types of problem that a bridge may encounter are:-

- A change on one side conflicts with a parallel change on the other side of a bridge ("Merge Conflicts").
- In the case of ClearCase, a reserved checkout on the same branch makes it impossible for a bridge to apply a change.
- A system error causes an update to fail.

Whilst some of these issues can be prevented by using an intelligent bridge setup (please see the Administration Guide for details), problems may still occur and you need to know how to resolve such problems.

As a general rule, **if you receive any error message from a bridge, you must inspect both ends of the bridge** and compare the file(s) in question in order to confirm whether the problem is permanent or not. The simple fact that you may not get any further error messages on future updates is not an indication that the problem has been resolved. Error messages from a bridge usually indicate that the bridge was unable to resolve an issue automatically and that **manual intervention is required**.

The following sections explain how to deal with the most common types of issues. Each section has a Signature subsection describing how to recognize a particular type of problem, and a Resolution subsection explaining how to resolve an issue of that type.

Note that if your Administrator has enabled INFO log messages for the CC2X Server, they can identify on which side of the bridge a problem occurred by the INFO message in the log file which preceded the error. INFO messages relating to SvnWorker relate to the Subversion side, CcWorker relates to the ClearCase side and GitWorker relates to the Git side.

3.1 Subversion File Merge Conflict

A Subversion merge conflict occurs if the same parts of the same file are changed at both ends of a bridge in between two update cycles. As update cycles tend to be fairly short, this should not be a very common occurrence. However, if you do receive a Subversion merge conflict error, you must act in order to resolve the problem.

3.1.1 Signature


A Subversion file merge conflict error looks as follows:-

```
[Bridge mybridge-1] *** CONFLICT! Merge errors detected in Subversion
workspace
The following files are in conflict (These problems must be resolved
through checkin of a fix version at either end of the bridge):
mydir/myfile
```

As the result of a merge conflict, CC2X commits the conflict including the Subversion diff, so a file with a conflict may look as follows:-

```
line1¶
line2¶
<<<<<<.mine¶
line3 ClearCase¶
=====¶
line3 Subversion¶
>>>>>> .r1234¶
line4¶
line5¶
```

The `.mine` side of the conflict shows the changes made in ClearCase, and the other side shows the changes in Subversion.

 When using a Subversion to Subversion Bridge, the `.mine` side of the conflict refers to the first repository defined in the Bridge Configuration (see screen shot below).

Subversion to Subversion Bridges

Change	Id	Name	Resolver Emails	Side	Repository	User Name	Password	Delete
				Subversion	.mine			
				Subversion				

To create a new bridge fill in the form above and click Apply.

From version 2.0 onwards, the CM Bridge performs updates (from Subversion) revision by revision. This style of update can cause multiple conflicts to occur on a single file (if the file has been committed more than once between each bridge update and an early commit creates a conflict). In this case, the signature of the conflict will look similar, however it will be slightly more complicated to resolve.

If this happens you have a complete history of the file commits in the Subversion repository and this will make resolving the conflict easier.

3.1.2 Resolution

The easiest way to resolve a Subversion merge conflict is to manually create a new revision/version of the file at either end of the bridge and committing/checking in the change. The bridge will then transfer the manually merged version of the file at the next regular update. The Subversion diff in the conflicted file gives you all the information you need to decide how to merge the changes and resolve the conflict.

3.2 Subversion Property Merge Conflict

In CM Bridge version 2.0, the functionality to bridge Subversion to Subversion repositories was added. This functionality introduced transferring of properties between repositories, and hence introduced the possibility of File/Folder property conflicts.

As for File Merge Conflicts, the Administrator will be informed via email that a conflict has occurred

3.2.1 Signature

A subversion property merge conflict looks as follows




```
[Bridge mybridge-1] *** CONFLICT! Merge errors detected in Subversion
workspace
The following file properties are in conflict:
prop2 on /mydir/file2
prop1 on /mydir/file1
Please see the clearvision:cc2x_property_conflict property for details.
Note: Please remove this property once the conflict has been resolved.
```

Note: When a property Merge conflict occurs, the CM Bridge will create a property on the file/folder where the conflict occurred. This property is called `clearvision:cc2x_property_conflict`

The `clearvision:cc2x_property_conflict` property will contain the conflict that occurred in the following format:

```
prop2>>>>value1-----value2<<<<
```


 `value1` refers to the property value in the 'first' repository defined in the Bridge Configuration (as in File Merge Conflicts above).

The original property (in this case `prop2`) will be updated (on both sides of the bridge) to reflect `value1`


3.2.2 Resolution

The easiest way to resolve this is to update the property on one side of the bridge and check in the change. The bridge will transfer the updated property at the next regular update.

It is recommended that you remove the `clearvision:cc2x_property_conflict` property once the conflict has been resolved.

 If the `clearvision:cc2x_property_conflict` property is not removed and another conflict occurs, it will be appended with the new conflict (effectively showing a history of conflicts) e.g.

```
prop2>>>>value1-----value2<<<<
prop2>>>>value3-----value1<<<<
```

 If you need this property to be different on either side of the bridge, you should request that the Bridge Administrator add this property to the list of ignored properties.

3.3 Subversion Lock Warning

A Subversion lock warning occurs if a file or directory have been locked by a user preventing the bridge from applying the changes from the other side.

3.3.1 Signature

A Subversion lock warning looks as follows:-

```
Cannot acquire subversion lock on 'afile': svn: warning: Path
'/repo/afile' is already locked by user 'fred' in filesystem
'/var/svn/repo/db'
```

3.3.2 Resolution

The easiest way to resolve a Subversion lock is to manually remove the lock by running `svn unlock --force /repo/afile` and applying the change manually.

3.4 ClearCase Merge Conflict

ClearCase merge conflicts are very rare by nature and you should not see these very often. A ClearCase merge conflict occurs if a file is checked out unreserved on the same branch as the bridges view and if the file is checked in between the bridge's checkout and checkin operations to apply a change. As the bridge only holds checkouts for a very short period of time, these types of conflicts do not occur very often.

3.4.1 Signature

A ClearCase merge conflict error looks as follows:-

```
[Bridge mybridge-1] *** Failed to propagate a change_file from
Subversion to Clearcase! (cc_file=mydir/myfile error=cleartool: Error:
The most recent version on branch "\main" is not the predecessor of this
version.␣
cleartool: Error: Unable to check in "mydir/myfile".)␣
```

3.4.2 Resolution

You can resolve a ClearCase merge conflict by merging the changes between the ClearCase version of the file at the ClearCase end with the Git/Subversion version at the other end and checking in a new version at either end of the bridge.

Please note that you can eliminate ClearCase merge conflicts by using a special branch for the bridge to use in ClearCase. Please refer to the CM Bridge Administration Guide for details.

3.5 Other ClearCase Errors

The most common problem is that a file is checked out on the same branch as the one that the bridge's view has been configured for. Other potential problems are system errors such as incorrect permissions or a vob or view having run out of disc space.

3.5.1 Signature

Examples of ClearCase errors are shown below:-

```
[Bridge mybridge-1] *** Failed to propagate a change_file from
Subversion to Clearcase! (cc_file=mydir/myfile error=
[Bridge mybridge-1] *** Failed to propagate an add_dir from Subversion
to Clearcase! (cc_dir=subdir/newdir error=cleartool: Error: Branch
"\main" of element is checked out reserved by view user1_view
("cchost1:c:\ClearCase_Stor
age\views\CLEARVIS-8D48C5\Administrator\user1_view").
cleartool: Error: Unable to check out "subdir".)
```

3.5.2 Resolution

The effect of this type of issue is that a change in Subversion has not been propagated to ClearCase. Before attempting to resend the data, you need to cancel the checkout in the other view. Please speak to the developer in question and inform him/her that his change conflicts with a change coming from the other end of the bridge.

Once the checkout has been cancelled, you can apply a property change on the Subversion side, which will cause the data to be resent, e.g.:-

```
svn propedit FORCE_SEND mydir/myfile
svn commit
```

This will cause the data to be resent on the next regular update. Note that the name of the property, FORCE_SEND in this case, is not relevant. Any property change will cause a file to be resent.

If the developer does not want to cancel their checkout, another possible resolution of this type of issue is to ask the developer to manually merge the changes from the Subversion side into their checked out version in ClearCase.

Please note that you can eliminate problems due to parallel checkouts by either of the following two methods:-

- Use a special branch for the bridge to import data onto, i.e. a branch not used by any other user. This setup is also commonly used for ClearCase's multi-site implementation.
- Always use unreserved checkouts. Note that you can enforce unreserved checkouts using a trigger that makes every checkout unreserved.

3.6 Other Subversion Errors

The most common problems you may encounter are related to a commit failing due to hooks preventing data from being checked in or the repository running out of available disc space.

3.6.1 Signature

Subversion errors typically look like the following examples:-

```
[Bridge newbridge-1] *** Cannot remove file 'nosuchfile': svn:
'nosuchfile' does not exist
[Bridge newbridge-1] *** Cannot commit changes to subversion:: svn:
Commit blocked by pre-commit hook (exit code 1) with no output.
```

3.6.2 Resolution

The easiest way to resolve a problem with applying changes to the Subversion repository is to reapply the change in ClearCase. If the problem was due to a system issue such as the repository running out of disc space, and if you only want to resend a file, you can make a *fake* change by checking the file out, and checking it back in again on the ClearCase side using the `-identical` option.

```
cleartool co myfile¶
cleartool ci -identical myfile¶
```

This will cause the file to be resent with the next regular update.

3.7 Git Merge Conflict

A Git merge conflict occurs if the same parts of the same file are changed at both ends of a bridge in between two update cycles. As update cycles tend to be fairly short, this should not be a very common occurrence. However, if you do receive a Git merge conflict error, you must act in order to resolve the problem.

Note that whilst most Git merge conflicts are detected, major errors such as running out of disc space during an update can cause the bridge to miss a merge conflict on the update cycle that immediately follows the major error condition. Whilst this theoretical problem is rarely seen in practice, it is recommended that you verify the results of the next update following a major Git update error.

3.7.1 Signature

A Git merge conflict error looks as follows:-

```
[Bridge mybridge-1] *** conflicts in workspace, committing the
conflicted files (mydir/myfile.txt, otherfile.txt) - THESE WILL REQUIRE
A MANUAL FIX ACTIVITY
```

As the result of a merge conflict, CC2X commits the conflict including the Git diff, so a file with a conflict may look as follows:-

```
one¶
<<<<<<< HEAD:myfile.txt¶
from ClearCase¶
=====¶
from Git¶
>>>>>> ca658604a5e414e6d1594d5ad2c7d99cd03cc3e3:file1¶
two¶
three¶
four¶
```

3.7.2 Resolution

The easiest way to resolve a Git merge conflict is to manually create a new revision/version of the file at either end of the bridge and committing/checking in the change. The bridge will then transfer the manually merged version of the file at the next regular update. The Git diff in the conflicted file gives you all the information you need to decide how to merge the changes and resolve the conflict.

3.8 Git Ahead Errors

3.8.1 Signature

The error message contains something like this:

```
# Your branch is ahead of 'origin/master' by 2 commits.
```

3.8.2 Resolution

This usually only happens after some other errors have occurred that the bridge could not automatically deal with. In this case, the administrator will have to step in and go in to the server's private repo for the bridge in the cc2x workspace (eg ~/cc2x_workspace/git/my_bridge).

Usually if you just do a

```
git push
```

from inside here, this will get the bridge back in step. (Do this while the server is stopped)

4 Background Information

The CM Bridge uses ClearCase attributes, Subversion revision properties and Git to track whether data was created by a user or by a bridge.

If you come across ClearCase attributes called `clearvision_cc2x` or Subversion revision properties called `clearvision:cc2x`, please do not modify this metadata unless instructed to do so by a Clearvision support engineer.