Using ObjecTime Developer 5.2.1 with ClearCase (v1.3)

Category:

Top

Miscellaneous

Purpose:

Top

This Tech Note explains the use of ClearCase with ObjecTime Developer 5.2.1.

It describes the steps required to properly configure your ObjecTime Session as well as ClearCase. It also describes a possible configuration management process that can be followed to ensure successful interaction of both tools.

Intended Audience:

Top

Project and configuration managers, ObjecTime developers, and loadbuilders.

This technical note assumes some familiarity with the ObjecTime Developer toolset and with configuration operations.

This technical note assumes some familiarity with ClearCase and its operation.

Applicable to:

Top

ObjecTime 5.2.1 and up

Description:

Top

Top

Table of Contents

Typographical Conventions

Desc TOC

This technical note uses the following typographical conventions:

- Source code and external commands appears in Courier Teal Text.
- Environment variables appear in Courier Bold.

- Tool entities appear in Times Bold.

Tool Configuration

Desc TOC

ClearCase

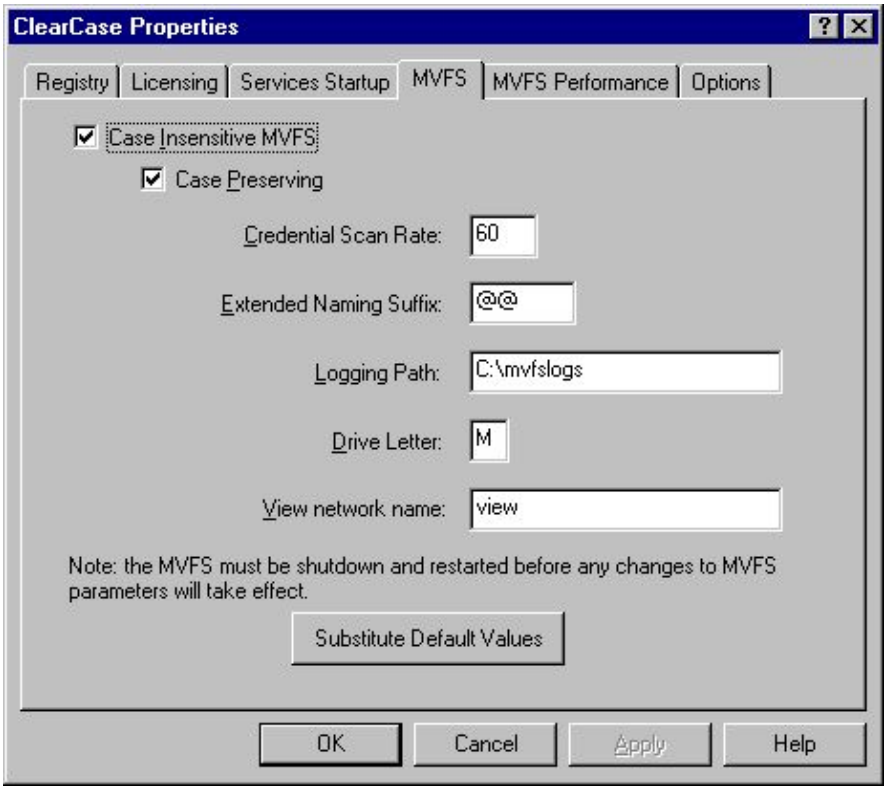
In general, the standard installation of ClearCase sets up the software correctly for use with ObjecTime Developer. There is, however, an exception in the case of the Windows NT installation. If you are using a UNIX installation, you can safely skip this section.

In the Windows NT case, the normal set up is to turn off case sensitivity on the MVFS drive. For ObjecTime Developer to work properly, the MVFS configuration of ClearCase must be set to be case preserving.

To turn on this capability, please do the following:

1. Either select the "Control Panel" button on the "Administration" tab of ClearCase Home Base, or select the "ClearCase" applet from the Windows NT control panel.
2. From the "ClearCase Properties" window that opens up, select the "MVFS" tab.
3. If it is not already done, enable "Case Preserving " by putting a check mark in the box left of it and click on the " Apply" button.

The following image shows the ClearCase Properties for these settings:



You are now ready to move on to the next step.

ObjecTime Developer

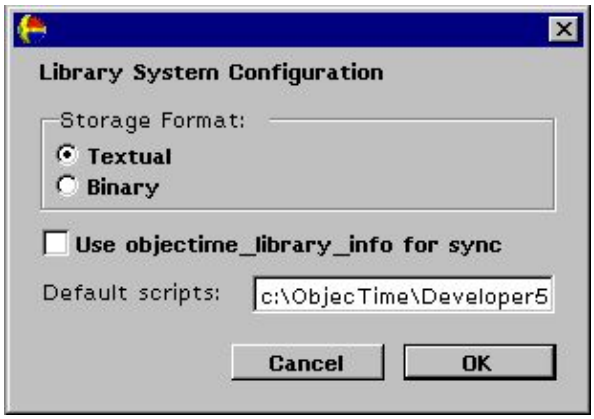
Desc TOC

You now need to configure your ObjecTime Developer session to use ClearCase.

First, we will need to configure the library interface to be able to use the ClearCase scripts. To do so, open the library configuration browser and verify the following:

- Storage format is set to "Textual"
- Do **NOT** "Use objectime_library_info for sync" (i.e., selection is not checked)
  Selecting this option will cause the toolset to use an internal method to determine whether a class requires synchronization. This method will generate false positives for most, if not all, classes/files. Using the supplied external synchronization script ensures that only classes that are actually different are synchronized.
- The default script location is set to the location of the ClearCase scripts.
  By default this is $OBJECTIME_HOME/bin/LibraryInterface/forClearCase (%OBJECTIME_HOME%/bin/LibraryInterface/for NT), but the desired location may have been set to some other directory by your project or configuration manager.

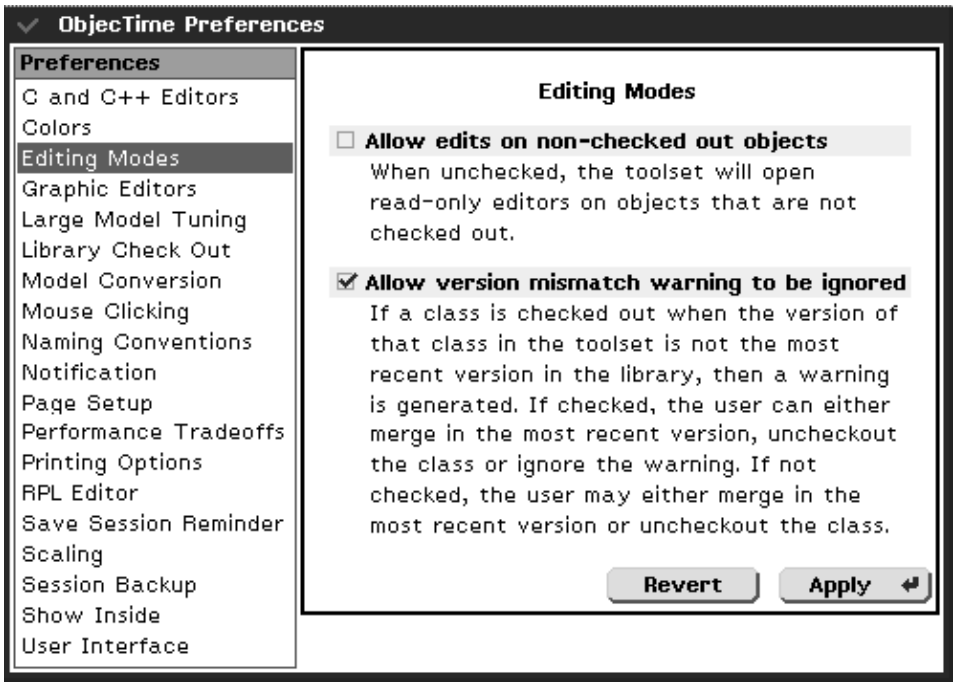
The following image shows the library configuration editor for these settings:



Next, we need to set up the user preferences correctly:

- Editing Modes:
  - Do **NOT** "Allow edits on non-checked out objects"
  - Do "Allow version mismatch warning to be ignored." This one is especially important when multi-branch development is being used.

The following image shows the preference editor for these settings:



Finally, if the "OBJECTIME_CODEGEN_TEMP" environment variable is used, it **must** be set to a different directory for each user. A directory on the user's local drive or in their own "home" directory is ideal. **Do not** point this environment to a directory in the VOB as it would create too many derived objects that are not required.

-

Note that this is the opposite to the usage currently described in the ObjecTime Developer 5.2.1 documentation. The usage described in this technical note should be considered correct.

Recommended Patch Lineup Desc TOC

You also need to make sure that the latest patches are applied. As a minimum, the following patches should be applied:

- Toolset patches lb01b and 001 through 020, inclusive. The correct order for these patches is: 001, lb01b, 002-020

Note that special patches such as those for ObjecTime Web Publisher and ObjecTime TestScope are not listed here. Please contact Customer Support for the correct lineup if you are using either of these tools.

- The new ClearCase scripts associated with patch 009 must be installed.
- Code generation patch 14.

Patches can be obtained from the ObjecTime Customer Support web site at http://www.objectime.com/support/restricted-dir/index.html

Desc TOC

ClearCase Concepts

In order to use ClearCase, the understanding of a few important concepts is essential. The following is not meant to be a complete explanation of the way ClearCase works (Rational Software has courses that explains this), but rather a quick overview of the concepts that are important in the context of this technical note. The concept of a VOB, of a branch, and of a view are the most important to review in order to understand the rest of this document.

VOB Desc TOC

A VOB (Versioned Object Base) is a repository of all the configured items and of all their version information.

Branch Desc TOC

A branch is a stream of development with a certain meaning. For example, you could define a development branch for a certain release or for fixing bugs. In a ClearCase context, the main branch is rarely used and then only for configuration management artifacts such as baselines and releases. Even those are sometimes relegated to other branches. All day to day development work is usually done on a development branch. This allows for parallel development as separate developers could be working on the same file, but in different branches representing different stages or phases of development.

The branch on which you are working is actually defined by your view's configuration specification.

View Desc TOC

A view is like a window into a VOB. By configuring the view, we can look at the contents of the VOB in different ways. This is accomplished through the view's configuration specification.

The view's configuration specification can be used to specify which files can be seen, which version of these files are seen, and to offer a certain level of security. For example, this configuration specification could restrict checkouts for a certain directory or branches.

It is beyond the scope of this document to explain the structure of a view's configuration specification. The reader should consult ClearCase documentation and possibly attend a ClearCase course to become more familiar with this.

Views are typically assigned to and used by a single developer and not shared. The reason for this is that they also contain some intermediate work files that could cause problems with development tools were they to be shared. Having a view used by one and only one developer eliminates this source of potential problems.

On Windows NT, a supplementary mechanism can be used to control and configure "standard" configuration specifications: View Profiles. These view profiles are part of a VOB and can be associated with a view. Once this is done, this view profile can be updated from a central location, making its maintenance a lot easier.

On UNIX, there is no equivalent to the view profile. Instead, it is recommended that you place the configuration specifications in text files that could then be read in to the view. These configuration

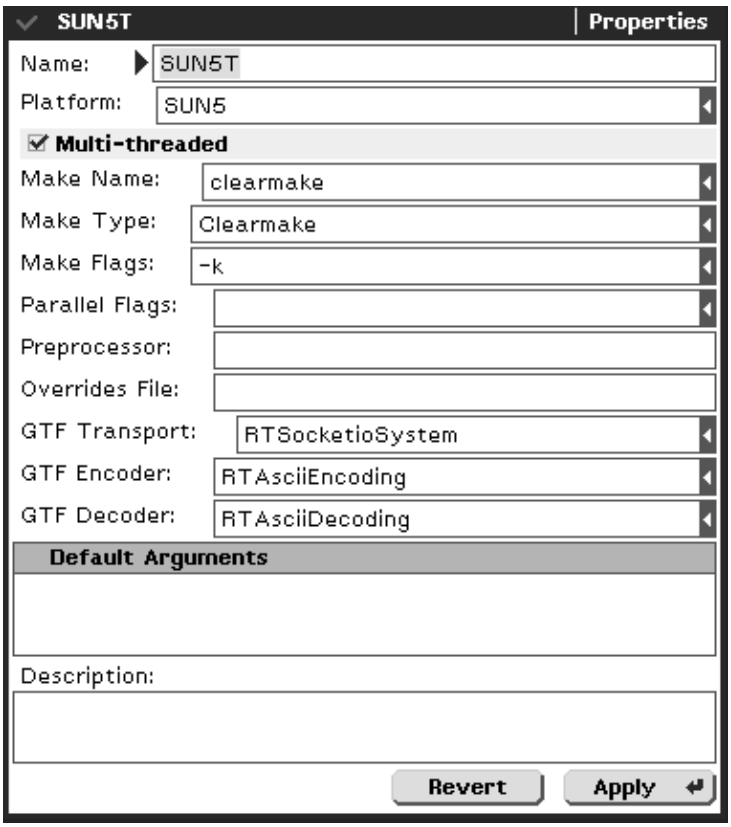
specification files could themselves be under control of ClearCase.

ObjecTime Update Configuration

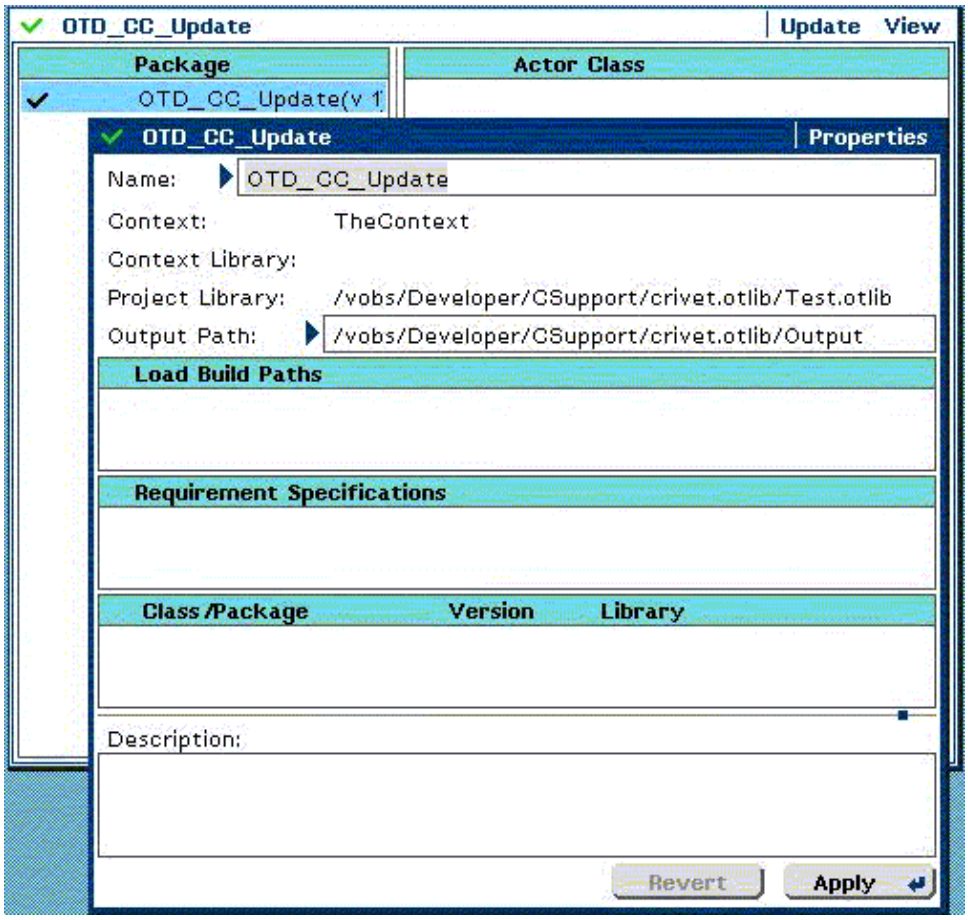
Desc TOC

It is recommended that you set up your packages before putting your model in the library. This will prevent you from having to check out the packages and project multiple time afterwards. Note that it is very important that classes only belong to one package.

If you are to use clearmake as your make utility, there is still some more work to be done. First, you will need to modify your update configuration(s) to use clearmake. This is done through the Language Option's Targets' properties editor. Both the "Make Name" and the " Make Type" must be set to clearmake (actually, clearmake and Clearmake, respectively). The following image shows these settings for a Sun Solaris installation:



Next, you have to change your update's properties to specify an output path. This output path is $PWD by default and must be changed to a path inside your VOB. It is suggested that you create a directory specially for this purpose in the VOB and put its path in here. Note that to allow for winkins, this directory must be versioned so that it is visible by all developers.



Note that if you changed the output path before setting your make type to use clearmake and you performed a build, linear form files will have been generated. These files will be found in an "LF" directory in the tree in the output path and will have to be removed (deleted) along with the "info" directory found in "C++/{Target|Simulation}RTS" before performing a full rebuild with the new configuration. If this is not done, these files will be used by clearmake instead of the correct classes, resulting in a non-changing executable.

ObjecTime Common Image

Desc TOC

At this point, especially if you are just starting development, you may want to create a common, pre-configured ObjecTime image that will be used by all members of the development team. You may also want to create a common, pre-configured context from the configured update you created in the previous section.

To create a context from your configured update, use the " New Context..." option from the update's menu and Give the context a significant name (e.g., "OurContext"). This will create a new entry in the Workspace browser. You can now delete your update and save and exit from ObjecTime Developer.

Now, let's make this session that you have just saved as the one used by everyone. To do this, first go to the $OBJECTIME_HOME/image (%OBJECTIME_HOME%\image) directory and make a copy of the ObjecTime5.2.otd file, to an "Original" directory for example. Now go back to your working directory and copy the image file you have just saved (also named ObjecTime5.2.otd) to the

$OBJECTIME_HOME/image (%OBJECTIME_HOME%\image) directory. From this point on, anyone creating a new working directory, either by using create_objectime_dir or through the ObjecTime Launcher, will be using this pre-configured image.

Note that anyone who already created a working directory prior to this should passivate their update, create a new working directory with the new image, and activate their update in their new session.

ObjecTime Model Configurations

Desc TOC

Although it was already good modeling practice to only work in packages within the ObjecTime Developer toolset, it is now essential when working with ClearCase as a configuration management tool.

Packages are the mechanism provided by ObjecTime Developer to help you organize your model. It allows you to easily define the layers and subsystems that are part of your architecture.

There is another ObjecTime Developer model entity that is very important when dealing with configuration management: the project. In order to really understand the reasons behind the need to use packages when dealing with configuration management, it would be a good idea to first explain what are projects and packages.

Projects and Packages

Desc TOC

Both projects and packages can be thought of as being simply containers or lists.

A package is a list of the all the classes (actor, protocol, and data) that it contains. Optionally, it can also have local configurations that override some of the properties of an associated project-level configuration.

A project is a list of the packages, configurations, and of any classes that are not part of any package. A project also contains thread information for your model. A project is created from an update by using the latter's "Check Out as Project File" menu option and then submitting it.

Packaging Heuristics

Desc TOC

Once projects and packages are understood, it is easier to determine some heuristics on their use when dealing with a configuration management tool.

- Projects must contain packages and configurations only.
- Packages are the only entities that can contain classes (actor, protocol, or data).
- Each class can belong to one and only one package.
- Packages can also contain package-level configurations and other packages (child or nested).
- Package dependency is one way only, i.e., no circular dependencies can be allowed between packages.

Directory Configuration (Non-Distributed)

Desc TOC

Once you have your model configuration done, at least at the main package level, it is easier to determine your directory tree. The easiest way to do it is to simply adopt the same "hierarchy" by having a sub-directory

for each package. In such a configuration, each sub-directory will be represented by a separate and distinct library in ObjecTime Developer. For example, you could end up with the following tree:

N:\VOBName\ProjectName\Package1.otlib
.....\SubPackage1.otlib
.....\PackageN.otlib
.....\Configurations.otlib
.....\Projects.otlib
.....\TestHarnesses.otlib
.....\Output

Where N is the mount point for your ClearCase view. Note that it is essential that this mount point be the same for all developers as this information is stored in the versioned object. Developers not having the same mount point would be unable to access the library.

The above example shows a Windows NT directory tree. A UNIX tree would be similar, except that the "\" would be replaced with "/" and the drive letter ("N:") would be replaced with the VOB mount point (e.g., "/vobs").

Note that this is only a suggestion. It is also possible to have multiple packages share the same directory (or library). However, this proposed organization allows for better control of the software.

Distribution

Desc TOC

It may sometimes be necessary to distribute development to different geographical locations (sites). If this is the case with your development effort, you will want to make sure that work can be isolated at a package level. This means that work done on a package (and its contained packages) should only be done at a single site.

Although the level of granularity for general configuration management within ObjecTime Developer is the class, the level of granularity when dealing with the distribution of the Configuration Management functionality is the package.

Note that this is not an absolute requirement, but rather a suggestion to reduce the amount of work required later on. If working on the same package is allowed for multiple sites, it will have to be done on separate branches, one for every site. This will then require merging of these branches later on, which is a manual process. See the sections titled " A Note on Merging" and "Merge two branches" later on in this document.

Directory Configuration (Distributed)

Desc TOC

First, let's list a few heuristics:

- VOB(s) are created at each location and replicated to the other locations.
- Each VOB is owned by its location and each location owns at least one VOB.
- Packages and their contained packages are assigned to a single VOB, depending on where the work is to be done.
- VOBs from remote locations are read-only. Developers should only be allowed to work on packages from their local VOB.

These suggestions are meant to facilitate and partition development.

So, instead of the directory structure suggested earlier, we would now have something similar to this (once again, using the NT notation):

```
N:\VOB_Common\ProjectName\Package1.otlib
.....................................\SubPackage1.otlib
.....................\PackageN.otlib
.....................\Configurations.otlib
.....................\Projects.otlib
.....................\TestHarnesses.otlib
N:\VOB_2\ProjectName\PackageN+1.otlib
.....................................\SubPackage1.otlib
.....................\PackageM.otlib
.....................\Projects.otlib
.....................\TestHarnesses.otlib
N:\VOB_X\ProjectName\PackageM+1.otlib
.....................................\SubPackage1.otlib
.....................\PackageL.otlib
.....................\Projects.otlib
.....................\TestHarnesses.otlib
N:\VOB_OUT\ProjectName\Output
```

Note that the view spans the multiple VOBs, and that all the VOBs share the same mount point on NT. The latter is especially important to ensure the drive letter stored in the versioned object is consistent.

You will also notice that there is an individual VOB setup for the output directory. This VOB should be the only one that is both readable and writable from all locations. This is the VOB that will contain all the generated files. As such, everyone needs read/write access to it in order to generate and share wink-in information.

Putting Your Model Under Configuration Control

Desc TOC

Now that your model and directories are organized correctly, you can check everything out and submit. There are two scenarios possible.

Single Library

Desc TOC

If you want the complete model in a single library (.otlib), you should check out the update as a project file and specify the desired library. You should then submit the update. The toolset will automatically ask whether you want to check out and submit all the other elements of your model. Answering in the affirmative will result in the complete model being stored in the library. Note that at this point, all elements are checked in.

Multiple Libraries

Desc TOC

If you have a need to store different model entities in different libraries, you should first make sure that the storage is correctly organized as packages. You will want the packages to be organized in such a way that each package's contents is located in a single library. You should then check out the desired elements to their

specific libraries. Once all elements are stored, you can then check out the update as a project file and all its configurations. Once everything is checked out, you can then submit. If anything was forgotten, you will be prompted to have these checked out.

You should now be ready to work and enjoy your development!

A Note on Renaming

Desc TOC

A common operation when working on a model is renaming model elements. This can occur for a variety of reasons such as making sure the name is a good descriptor of the functionality provided. Unfortunately, there is no easy way to do this from within ObjecTime Developer whilst keeping historical information about the model element.

The easiest way to rename a model element is to change its name while the element is not checked out. This will effectively create a new model element having the same characteristics, but a different name, and its historical information will therefore be lost. This new model element should then be checked out along with all other model elements that refer to it. One such model element would be its container package. Once all affected elements are checked out, the update can then be submitted.

By using a combination of ClearCase commands and ObjecTime Developer toolset functionality, it is also possible to keep the model element's historical information. This process is described later in a suggested development process.

A Note on Merging

Desc TOC

ClearCase offers tools to automatically (or manually) merge the elements of two branches. These tools were designed to work with simple text files and as such may not be suitable for merging ObjecTime Developer files.

Although we have previously specified that we want the versioned objects to be stored in textual format, they still contain graphical information. Because of this, letting the ClearCase tools, such as findmerge or xclearmerge, perform the merge operation may result in unviable linear form files that will not load in the toolset.

As such, we recommend the use of the **Class Version Merge** facility provided by the toolset in order to perform the merge of classes. The usage of any other tool should be considered as *unsupported and at your own risk.*

Version in view vs. Latest Version

Desc TOC

When merging an element into your model, you are presented with a dialog that allows you to either merge with the version in view or with the latest version. When using ClearCase, you should always be selecting to merge the version in view. You should therefore always set your view to select the versions you actually wish to see.

Selecting to merge the latest version can result in unexpected behaviour as the toolset will actually scan all branches to determine which is the most recent (timewise) version. You may therefore end up with unstable versions from someone else's development branch!

To Freeze or Not To Freeze

Desc TOC

One question that often occur is when to use the "Freeze versions in Library" checkbox when activating/merging a class into the update. In general, you should rely on ClearCase to provide you with the correct version and never use this option.

The only safe use would be to look (and only look) at a previously stored version of a project. In any case, the project loaded will be the one in view, but the package versions will be the ones specified in the project file. You could use this feature, for example, to determine the differences between the last baseline version of the project with the work that has been done since (and that is currently in view).

Suggested Development Process

Desc TOC

Here is an example of a development process that can be used when working with ObjecTime Developer and ClearCase. Before beginning, it is useful to define a few global rules:

- Each developer has their own view and views are not to be shared.
- A global code generation and build directory must be defined for all developers to use. This directory **must** be in a VOB. Note that there is no danger of developers interfering with each other as each view maintains its own storage area. This is required for wink-ins to occur.
  See sections " Directory Configuration (Non-Distributed)" and " Directory Configuration (Distributed)" for recommended directory configuration.

The following sections describe some of the usual processes that are performed by different persons in the project.

Developer Processes

Desc TOC

Work on existing classes

1. Set configuration specification to the correct view profile for the component on which you are working.
2. If this was not done previously, open the last baseline project to create an update in your workspace.
3. If the update already exists, then perform a synch with library operation to bring in any new files that may be available.
   This operation can also be done during the day to check for and bring in any changes done by other teams.
4. Check out classes to be worked on.
5. Do work.
6. Save to library.
   Note that this is performed automatically by the "compile" toolset command. Also note that this does not check in the changes you have made - it only creates a view-private file of your changes.
7. Test.
8. If everything OK, submit changes (and possibly check out and submit the packages and/or project).

Desc TOC

Add a class to a package

1. Start with a running ObjecTime Developer session.
2. Check out the package to which you want to add a class.
3. Make sure the package is selected in the package pane and create the new class.
4. Checkout the new class from the same library as the package.
5. Make any necessary change to the class. You can also check out other classes to add references to the new class.
6. Once you are satisfied, check in all the classes that are checked out in the package.
7. Check in the package.

Remove a class from a package and model

Desc TOC

1. Start with a running ObjecTime Developer session.
2. Check out the package from which you want to remove a class.
3. Select the desired class and delete it with children. This will remove the class from your model.
4. Check in the package.

Move a class between packages

Desc TOC

Note that this method will leave the class' disk file in the same location. If your process requires that the disk file also be moved, please look at the next process.

1. Start with a running ObjecTime Developer session.
2. Check out both the source and destination packages.
3. Select the desired class and open its properties dialog. Add the destination package to the package list and remove the source package.
4. Check in both packages.

Rename a class

Desc TOC

Note that this process can also be used to change a class' disk file location.

1. From a dos prompt, enter the commands
   cleartools co .
   cleartool mv <oldclass> <newclass>
2. In ObjecTime, make sure the "Allow edits ..." checkbox is checked in the "Editing modes" preference dialog.
3. Open the properties of the class in question and change the name to the new class name.
4. Perform a Sync with Library for that class.
5. When it merges in the class, make sure the "Merge latest ..." is **NOT** checked.

Integrator Processes

Desc TOC

Create a new project

1. Set configuration specification to the correct view profile for the component on which you are working.
2. Create a new update in your ObjecTime Developer workspace, based on "TheContext". Name this update something significant, i.e., do not keep the suggested name ("TheContextUpdate").
3. Merge in common packages (e.g., CommLayer) from the library.
4. Merge in the desired component packages, making sure that dependant packages are merged in first.
5. Merge in the configurations for your component.
6. Activate one of these new configurations and delete the one that was automatically created when the update was created. This will prevent you from using a non-versioned configuration.
7. Set the output path in the update's properties to a directory in the VOB. This directory should versioned and common to all developers.
8. If necessary, set the thread information for the project.
9. Check out and submit the project file to CM.
10. You now have a new project from which to work.

Add a package to a project

Desc TOC

1. Start with a running ObjecTime Developer session.
2. Check out the update as a project file.
3. Create a new package in the package list.
4. Checkout the new package from its library.
5. If required, add classes to this new package.
6. Submit the update (project).

Remove a package and its contents from a project

Desc TOC

Note that since packages represent logical groupings, you should never have to remove a package without also removing its contents.

1. Start with a running ObjecTime Developer session.
2. Check out the update as a project file.
3. Remove from the model all the classes that are contained within the package that is to be removed. *Be careful when working with child packages!*
4. Remove the package from the model.
5. Submit the update.

Merge two branches - Method 1

Desc TOC

This process is usually done when a new "baseline" is to be established for the loadbuilder. Using this new merged branch, the loadbuilder can proceed with daily/weekly loadbuilds.

1. Set your view's configuration specification (config spec) to the branch to which you want to merge (destination branch).
2. Start ObjecTime Developer, load the project or sync with library to get the version in view.

3. Check out the classes that will be modified (you can use ClearCase findmerge to determine which classes need merging - **DO NOT** let it do the merge).
   If new model elements are to be added, you must also check out their container (e.g., package).
4. Set your view's config spec to the branch from which you want to merge (source branch).
5. Refresh the library browser in your ObjecTime Developer session to show the new class lineup.
6. Merge in any **new** model elements by dragging them from the library browser into the model.
7. For any **existing** model element, use ObjecTime Developer's Class Version Merge tool to merge the classes in the update with the classes from the library browser. The class in the update must be the Original and the one from the library the Variant for the changes to be propagated.
8. Set your view's config spec back to the branch to which you want to merge (destination branch).
9. Refresh the library browser in your ObjecTime Developer session to show the new class lineup.
10. Submit the update.

Merge two branches - Method 2

Desc TOC

This is an alternate method that can be used when the branch and version information is known for all elements to be merged.

1. Set your view's configuration specification (config spec) to the branch to which you want to merge (destination branch).
2. Determine the branch and version information for all model elements to be merged.
   You can use ClearCase findmerge to determine which classes need merging, along with their branches and versions, but **DO NOT** let it do the merge if the class already exists in the destination branch! You can **only** findmerge to add new model elements to the destination branch.
3. Start ObjecTime Developer, load the project or sync with library to get the version in view.
4. Check out the classes that will be merged.
5. Merge in any new class that has been added (their container should already be checked out as it is a modified model element).
6. Use ObjecTime Developer's Class Version Merge tool to merge the classes in the update with the correct branch and version of the classes from their version browser (accessible by double-clicking on the model element in the library browser). The class in the update must be the Original and the one from the library the Variant for the changes to be propagated.
7. Submit the update.

Loadbuilder Processes

Desc TOC

Build the model - from Toolset

1. Make sure the integrator created the new lineup and set your view to the correct config spec.
2. Load the project or sync with library to ensure you have the correct lineup.
3. Check out the update as a project file.
4. Check out every package.
5. Submit the update.
6. Build the model.
7. If the build is successful, advise the developers of the availability of a new loadbuild baseline.

Checking out the project and the packages will ensure that the lineup is saved in ClearCase as part of these

files' information.

Build the model - from Command Line

Desc TOC

1. Make sure the integrator created the new lineup and set your view to the correct config spec.
2. Use the command required to build the desired project.
   This command can be extracted from the "compile.output." file after a build from the toolset. This is the perl command shown on the second line, after "# This file generated by:". The loadbuilder can save this command and put it either in a script (shell, batch) or a makefile to be run at a later time. Alternately, you can follow the instructions found on page 89 of the ObjecTime Developer 5.2.1 Getting Started Guide Release Notice.
3. If the build is successful, advise the developers of the availability of a new loadbuild baseline.

Configuration Manager Processes

Desc TOC

Create a baseline

1. Use ClearCase findmerge to determine if any new classes have been added.
2. Use ClearCase findmerge to merge any and all directory changes.
3. Set the view's configuration specification (config spec) to point to your baselining branch (e.g., "/main/LATEST").
4. Start ObjecTime Developer.
5. Make sure the ObjecTime Developer configuration allows edits on files that are not checked out.
6. Load the last baseline project into OTD.
7. Set the view's config spec to view the modified packages.
8. Refresh the library browser in your ObjecTime Developer session to show the new class lineup.
9. Merge in the package(s) from the library browser.
10. Repeat last three steps for other packages or branches.
11. Set the config spec back to your baselining branch (e.g., "/main/LATEST").
12. Refresh the library browser in your ObjecTime Developer session to show the new class lineup.
13. Check out all the modified elements (from the "findmerge" list done previously or recognizable by branch information in the version) and their containers (packages).
14. If a creating a new baseline, check out the update as a project file.
15. Submit the update (project).
16. Exit ObjecTime Developer.
17. Using ClearCase tools, apply a new label to the new baseline.

Create a baseline - Copy-Merge

Desc TOC

The method described below assumes the existence of a defect tracking software that identifies the file version to be merged back into the baseline. It also assumes that there is no parallel development on the source branch(es). If a file (class) is modified on more than one branch, then this method should not be used and the classes should first be merged manually using the ObjecTime Developer Toolset.

1. Set your view to see the baselining branch (e.g., "/main/LATEST").
2. From a defect tracking system (or findmerge - but do not let findmerge do the merge), get the files and version that need to be copy-merged.

3. Create a script (shell or Windows batch) to checkout the files, obtained in the previous step, from the current (destination) branch.
4. The above script should then copy the desired file/version using ClearCase extended path names (i.e., path names that not only contain the file name and location, but also its branch and version information.
5. The script should then check in all these files.
6. Start ObjecTime Developer and load project or sync it with library.
7. Check out the update as a project file.
8. Check out every package.
9. Submit the changes.
10. Exit from ObjecTime Developer.
11. Apply new label to new baseline.

Limitations:

Top

None

See also:

Top

- "ObjecTime Developer 5.2.1 Getting Started Guide and Release Notice"; Chapter 8 - ClearCase Support Enhancements; ObjecTime Limited part number OT-R521-DOC808, Version 1.0, February 1999.
- "ObjecTime Developer 5.2 User Guide"; Chapter 9 - Differences and Class Version Merge Tools; ObjecTime Limited part number OT-R520-DOC801, Version 1.0, August 1998.
- "Organizing and Managing Software Projects Using ClearCase on Windows"; Rational Software white paper available as part of the Rational Unified Process.

This technical notes is also available as a PDF file.

What's New?

Top

- v1.0 -- 1999.08.13
  - ♦ Original version.
- v1.1 -- 1999.09.27
  - ♦ Added instructions for getting build command line.
  - ♦ Corrected baseline creation (without copy-merge) and merge instructions -- no need to leave the toolset.
  - ♦ Added note on renaming.
  - ♦ Added second merging method to use the class browser.
- v1.2 -- 1999.10.13
  - ♦ Added instructions on renaming model elements.
  - ♦ Made some minor formatting changes.
- v1.3 -- 1999.10.20

- Corrected recommended lineup for ObjecTime Developer patches.
- Corrected "*A Note on Renaming" to reflect the renaming instructions added in the last revision.*