WORKING IN TEAMS WITH CASECOMPLETE® AND PURECM

Contents

Working in Teams with CaseComplete

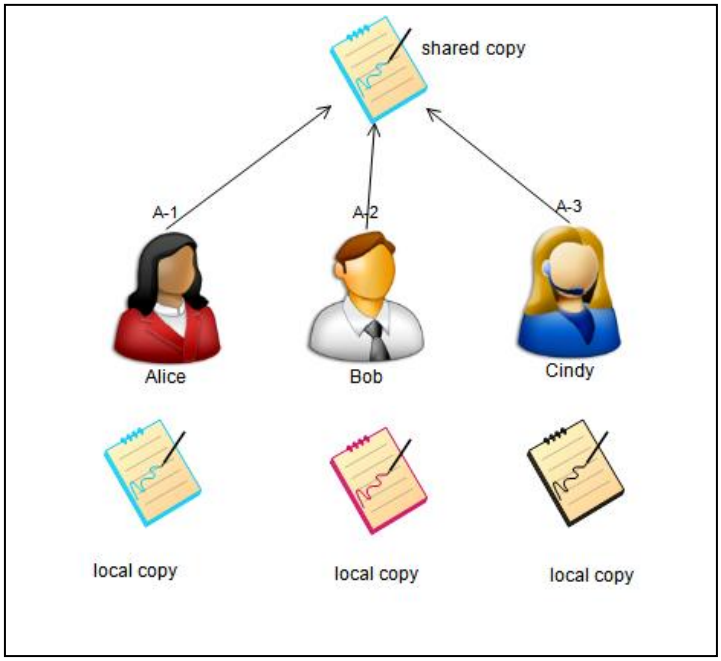
This guide will help you get started using CaseComplete 2009R2 in a team environment. This version of CaseComplete integrates with version control systems to allow members of your team to coordinate your CaseComplete work – even if you are in different parts of the world.

Need an introduction to how version control works?

If you are already familiar with version control systems, you can probably skip this section. If it's been a while since you've worked with one, or you never have, you may pick up a tip or two in this section.

The basic idea behind version control systems is to be able to have multiple people working on related files in a manner that can prevent them from stepping on each other's toes and to keep a running history of changes made. Consider the following scenario:

Alice, Bob and Cindy are responsible for creating separate sections of a document. They have decided to store the master copy of the document in a shared folder on their local area network, and have agreed that they will copy that master copy to their local hard drive, make the changes and copy the changed version back to the shared folder. The newly copied version then becomes the new master copy.



Unfortunately, they didn't work out a schedule for coordinating who would be changing the master copy at any particular time. All of them made a copy right away and started working. Bob finished first and copied his version back to the shared folder. Cindy finished a few minutes later and copied her version. Later in the day Alice finished her work and copied her version.

Since Alice was the last person to copy the file, her version overwrites Bob and Cindy's version. Their work is lost, and there is no way to get back those earlier versions created by Bob and Cindy (unless, of course, they've saved a copy someplace safe).

Version control systems provide a means of coordinating changes so this kind of scenario doesn't happen. The version control system tracks each change made to a file in a coordinated manner so that changes aren't lost by overwriting. Since each set of changes is tracked, you can also use the version system to go back to an earlier version of the file.

Version control systems maintain a database of project files in a "repository" and allow individual users to make a copy of those project files in "working folders." The repository is updated by the version

control system; users work on the copies in their individual working folders and submit changes to the version control system which in turn updates the repository. Throughout this document, the term "repository" will refer to the central location where the version control system maintains project information, and "working folder" will refer to the copies of project files that are modified by the individual user.

There are two primary types of file control with a version control system: exclusive checkout and multiple checkout. Each type of control has advantages and drawbacks.

Exclusive Checkout

Exclusive checkouts control changes to the file by restricting the ability to update a file to one person at a time. To make a change to a file, the file must first be "checked out." This locks the file, and no one else can check out the file while it remains locked. When changes to the file are complete, the file is checked back in and the lock is removed. The changed version then becomes the "current" version. The next person to check out the file will get that latest version and will have exclusive rights to update the file until the lock is removed. Note that a checkout can be canceled – in that case the file remains unchanged and the lock is removed.

The advantage to exclusive locks is that updates to the file happen one-at-a-time, and it is relatively straight forward to coordinate those sequential changes. The disadvantage is that only one person can make changes to a single file at a time. Two people can't be working on the same file simultaneously.

Multiple Checkout

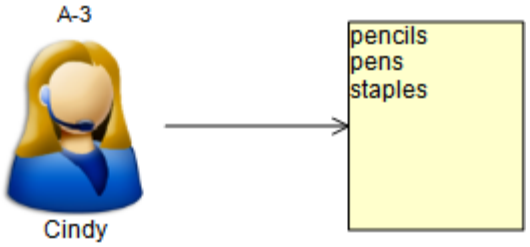
Multiple checkout addresses that problem by using a different approach. Instead of using a lock-update-unlock approach where access is limited to one person at a time, this approach permits more than one person to make a copy of the file, make changes and then merge those changes with the then-current version to create a new version.

In order for multiple checkout to work, the version control system needs to be able to figure out what has changed between the current version of a file and the version that you started with and the one containing the changes you've made. Version control systems use a differencing algorithm to find what has been changed, and what changes need to be applied to make the original identical to the new version (and vice versa, so the previous version can be recovered once the new version is committed). Differencing algorithms have their limitations though. First, they really only work well with text files. Binary files – for example, an image – just don't lend themselves well to differencing in the context of a change management system. Secondly, they can't always figure out the right thing to do when sets of changes could result in more than one outcome. That particular circumstance is called a "merge conflict."

Merge Conflicts

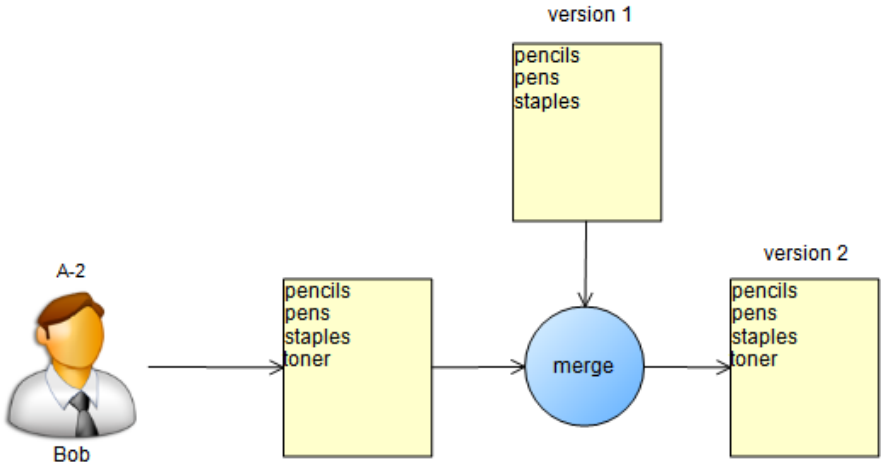
Merge conflict happens when a set of changes could have more than one result, and the version control system has to ask a human being for help to sort things out. Here's an example of a merge conflict:

Cindy creates a text document containing a shopping list of office supplies:

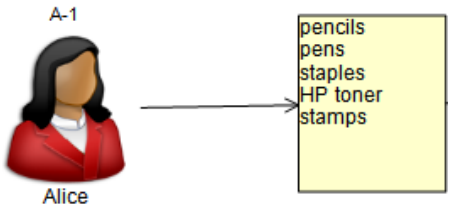


She commits that file to the version control system and asks Bob and Alice to review and edit the list.

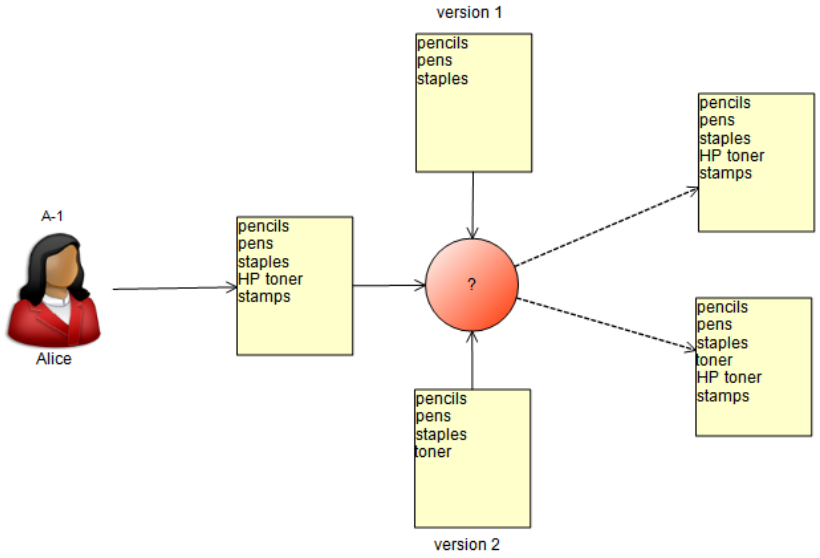
Alice and Bob both get a copy of the list from Cindy's version (let's call that "version 1"). Bob makes a change and commits that to the version control system. The version control system compares Bob's changed version against the version he started with (version 1). Since version 1 is still the most current version, it's easy to merge Bob's changes in.



Not long after Bob commits his changes, Alice finishes her changes and tries to commit them. She started with version 1, and has added two items to the list:



Before her changes can be committed, however, her working copy (which is based on version 1) needs to be merged with the newest version (version 2). The version control system cannot reconcile the changes because there is more than one possible result of the changes:



The result is a merge conflict that has to be resolved by a human being who can figure out whether the resulting version 3 should include or exclude "toner."

Merge conflicts are important to understand if you are using an external version control system that allows multiple checkouts. The files that CaseComplete reads and writes are a little more complicated than plain text files. Those files contain information formatted as standard XML (eXtensible Markup Language). If you permit multiple checkouts, you may encounter situations where your version control system cannot reconcile a set of changes. In that case you will have to manually work with the XML data. If you are not familiar with XML, it's best to not permit multiple checkouts.

Note: If you are using CaseComplete's version control addin that's included with CaseComplete, you don't have to worry about merge conflicts. The CaseComplete Shared Project addin's version control uses an exclusive locking strategy so that all changes are made sequentially, and the CaseComplete files are locked while they are being updated.

CaseComplete Files

Version control systems operate on files, so it's important to know a little about the files that CaseComplete uses. This will help you to effectively structure your CaseComplete projects and optimize the way your team works.

CaseComplete creates several different files as you work with it.

<project>.ucd	main project file
<project>.ucc	Stores tracked changes
<project>_glossary.ucg	glossary file
<project>.ucd.autosave	autosave file, automatically backs up your work periodically
<diagramname>.ndx	diagram file
<packagename>.ucp	package file

<projectname>.settings	Settings file
<diagramname>.ndx.autosave	autosave diagram file, automatically backs up your diagrams periodically

The project file, package files, glossary files and diagram files all contain project information that needs to be shared with other people working on the project. In order for those files to be shared, CaseComplete needs those files to be managed under version control. The autosave and settings files, on the other hand, don't contain shared project information – they are specific to you and aren't version controlled.

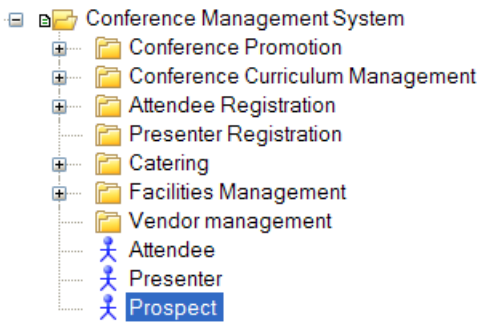
Note: CaseComplete allows you to separately control (check out, check in and track versions) of the following kinds of project elements:

- Packages
- Glossaries
- Diagrams
- Change tracking files

Package Files

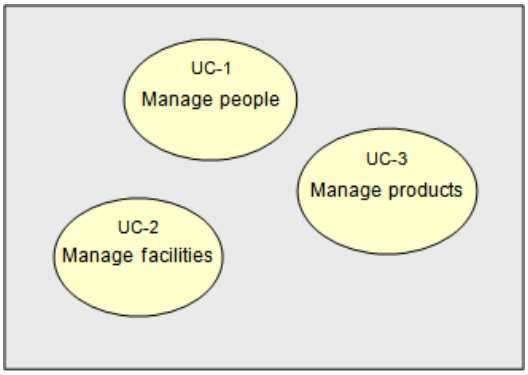
By default, all of the use cases, actors and requirements you create in CaseComplete are stored in the main project file (the <project>.ucd file). If you are the only person working on a CaseComplete project at any given time, that doesn't present any particular problems. You can check out the file to lock it, make updates, and check it back in when you're done. However, if your version control system is using exclusive locking (for example, the CaseComplete Shared Project addin), no one else can make changes to that project while it is checked out – everything is in a single file. The answer is to partition your CaseComplete project into separate package files.

CaseComplete allows you to logically partition your work into separate packages. Packages are like folders that allow you to organize the things in your project. For example, you may want to organize your project so requirements and use cases are segregated by the area of business concern.

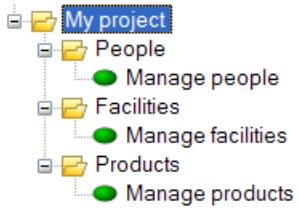


Each package is initially created in the same file as the main project. You can, however, save a package (and its contents) as a separate file. By creating specific packages in separate files, those parts of your project can be checked out and modified separately from the main model file.

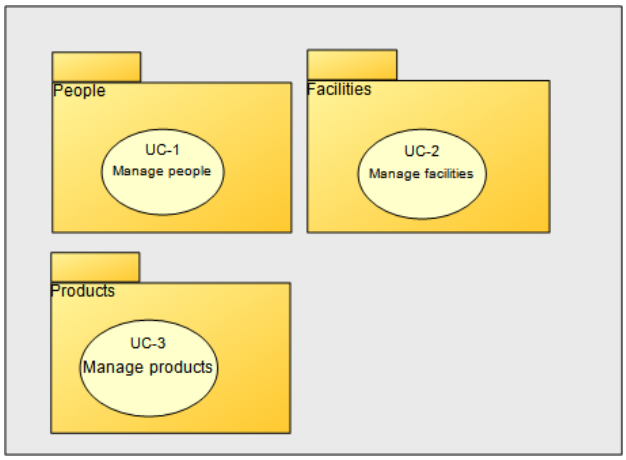
For example, suppose that a project contains three use cases. All three use cases are stored in the main project file.



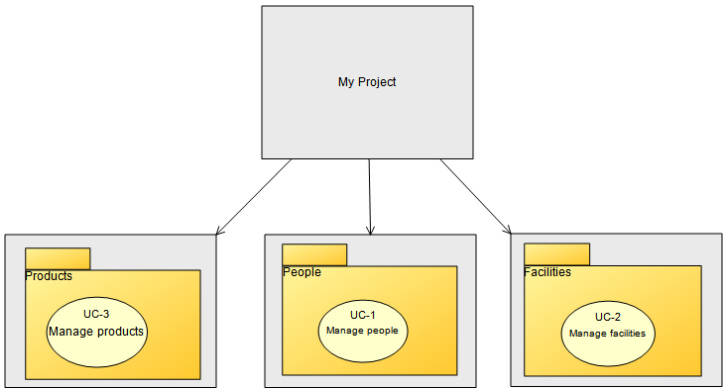
In an exclusive locking environment, the project file has to be checked out before a change to any of the use cases can be made. If more than one person needs to work on the project at the same time, start by thinking about how responsibilities for updating the project are going to divided amongst the team members. In the simple case above, the division of work is likely to be based on people, facilities and products. Start by creating a separate package for each of those three areas. Then organize your model by creating or moving the use cases, actors and requirements that are relevant to each of those areas into its respective package:



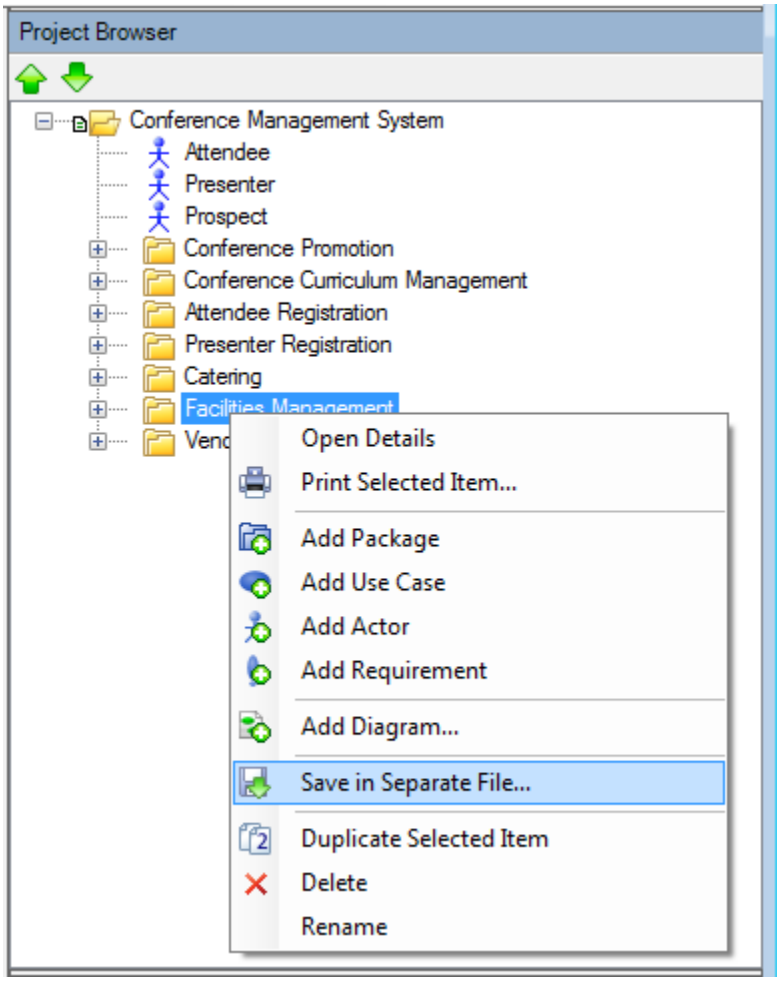
This logically breaks the model into three areas of interest. The information for each package is still stored in the main project file, though.



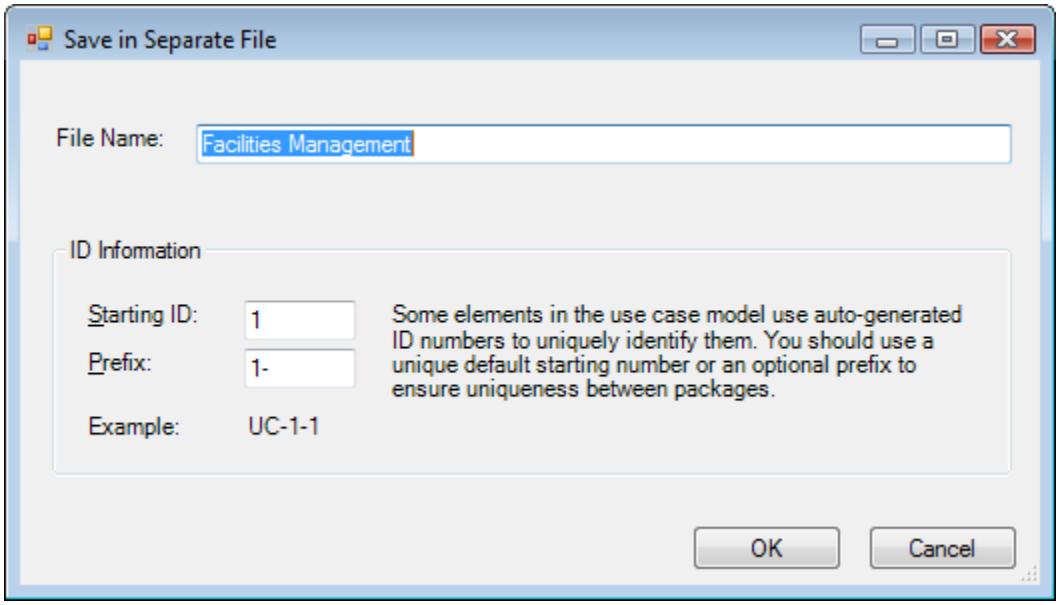
If each of these three primary packages is made into a separate file, then each one can be separately checked out without locking up the other packages. The main project file still ties all of the packages together with references, but the information for each of these packages is stored in a separate file:



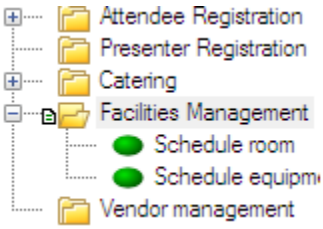
To save a package in a separate file, simply right-mouse-click on the package you want to save, and select “Save in separate file.”



You'll be prompted to provide a filename for that package file, and optionally add an identifier prefix for the elements in the package:



After you have successfully saved the package as a separate file, the icon in the model explorer for that package will have a decoration indicating that it is maintained in a separate file.



Structuring and Partitioning your Project

Deciding how to structure your project into packages will take some thought. Start by thinking about how the requirements responsibilities will be divided and how the logical areas of interest are divided. You probably won't want to structure things solely around work breakdown – having packages named for each of your team members might make sense in the short term but will quickly lose coherence over time. Instead, use the areas of responsibility and interest to guide your structure.

Be careful about the granularity of your project structure. If you don't partition the project sufficiently, your team members will be contending for checkouts. On the other hand, not every package needs to be stored in a separate file and version controlled separately. Too many "moving parts" can make things overly complex. Fortunately, CaseComplete makes it easy to refactor your project and to move things around to tailor the project to your team's needs.

What's Next?

The remaining sections of this guide provide guidance to start working in teams with CaseComplete.

Getting Started with CaseComplete and PureCM

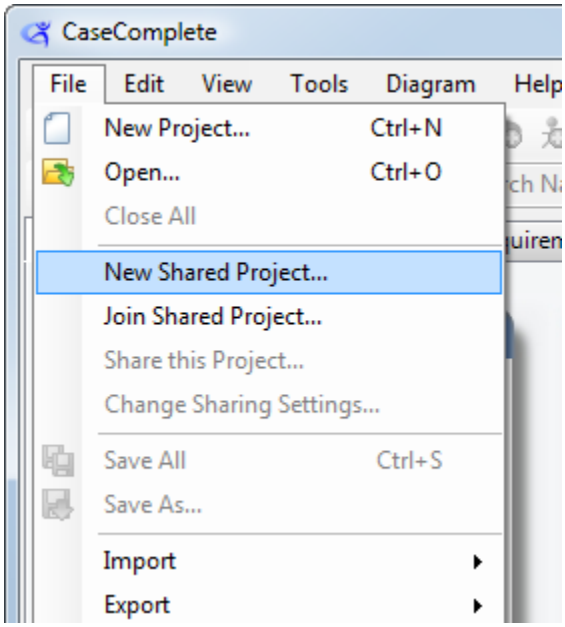
CaseComplete allows you to use PureCM for managing your CaseComplete projects. Using this option, CaseComplete will manage check out, check in and versioning through PureCM.

Note: If you happen to have more than one MSSCCI-compatible SCM system installed on your computer, please realize that only one can be registered as your MSSCCI provider at a time. Refer to your vendor's documentation if you will need to switch out providers. If you only have one SCM installed on your computer, CaseComplete will automatically recognize it.

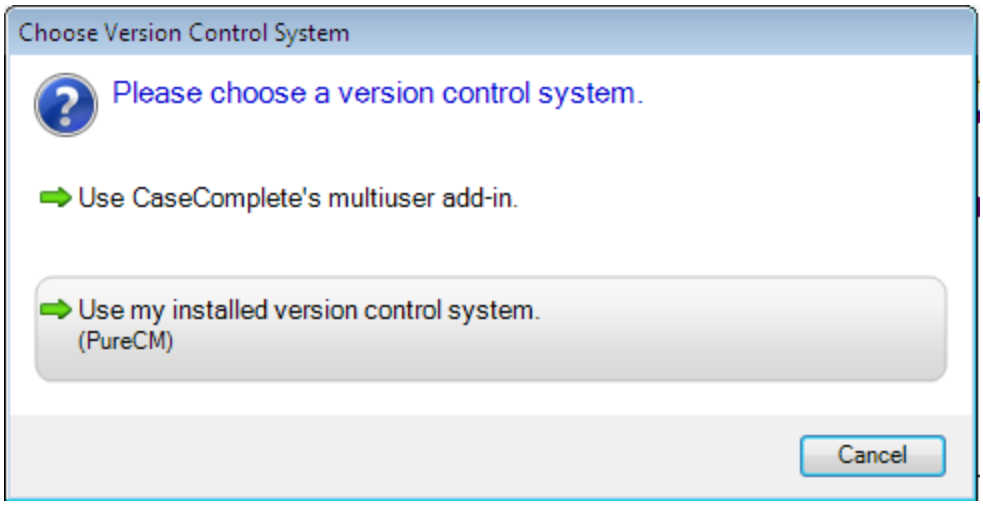
Create a New Shared CaseComplete Project

In order to create a shared project, you'll first need to identify or create a PureCM repository, then create a workspace for a stream in that repository. Use the PureCM client to perform these steps. Depending on security policies and your privileges in PureCM, you may need to have an administrator perform these tasks.

Start CaseComplete. Select New Shared Project from the File menu:

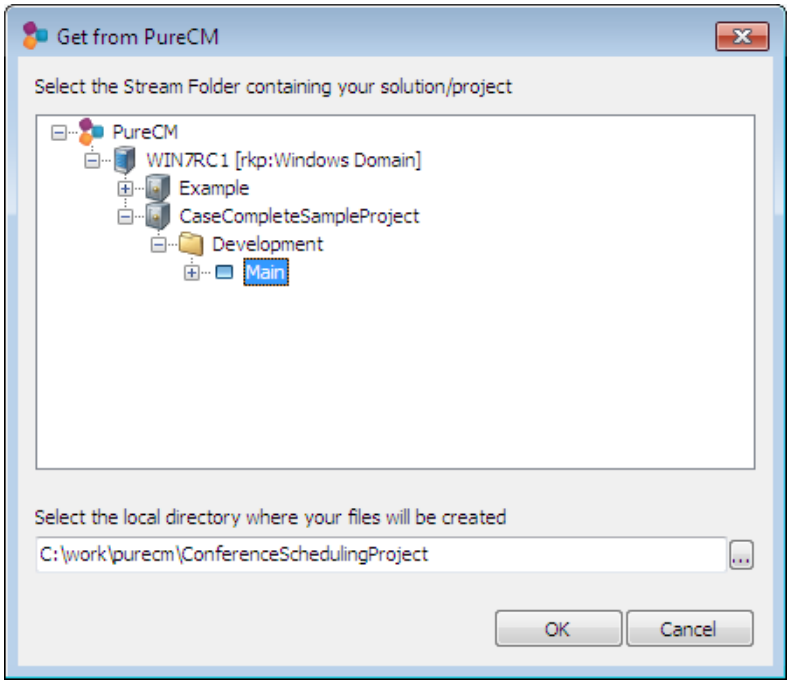


Next, select "Share This Project" from the CaseComplete File menu. CaseComplete will ask you which version control system to use. Select "Use my installed version control system":

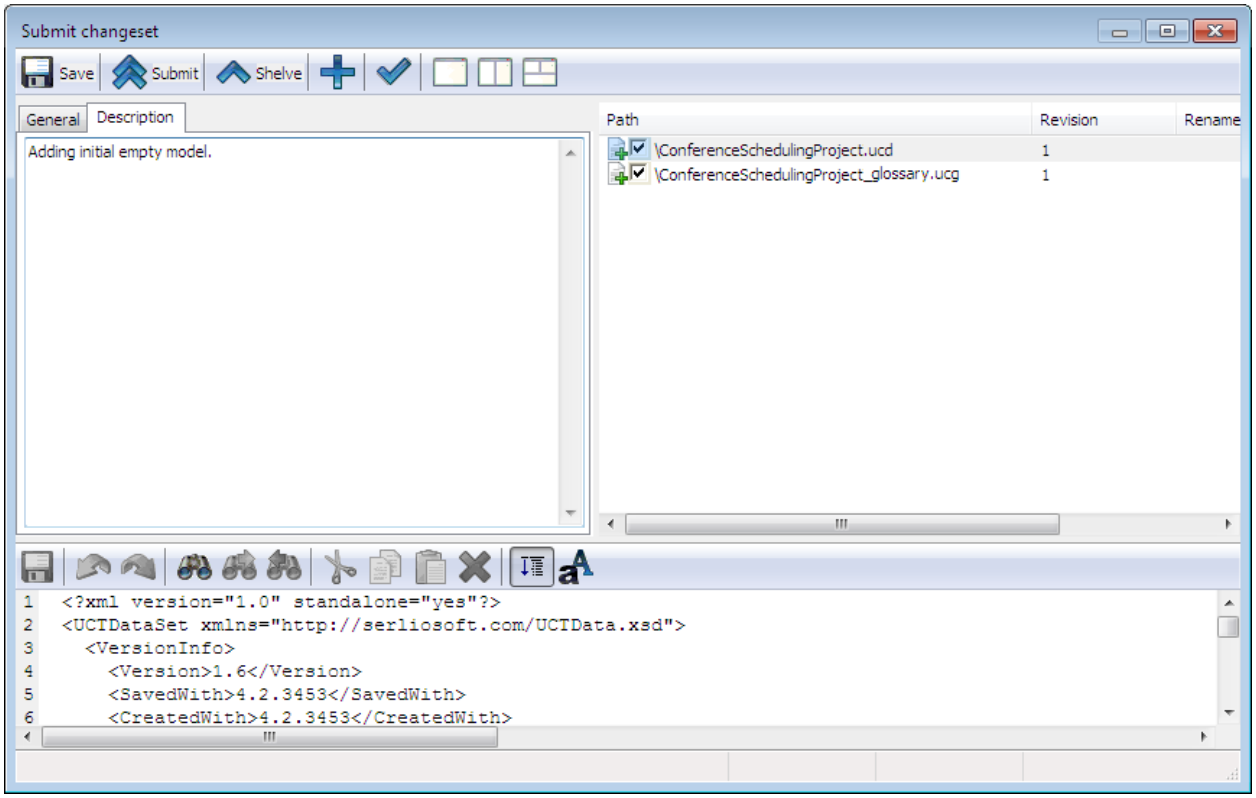


A PureCM dialog will display and allow you to confirm the stream and workspace associated with this project. Click Ok.

Note: You must select a local directory that is a workspace created with PureCM



Next, PureCM will confirm your submission and allow you to annotate the checkin with comments:

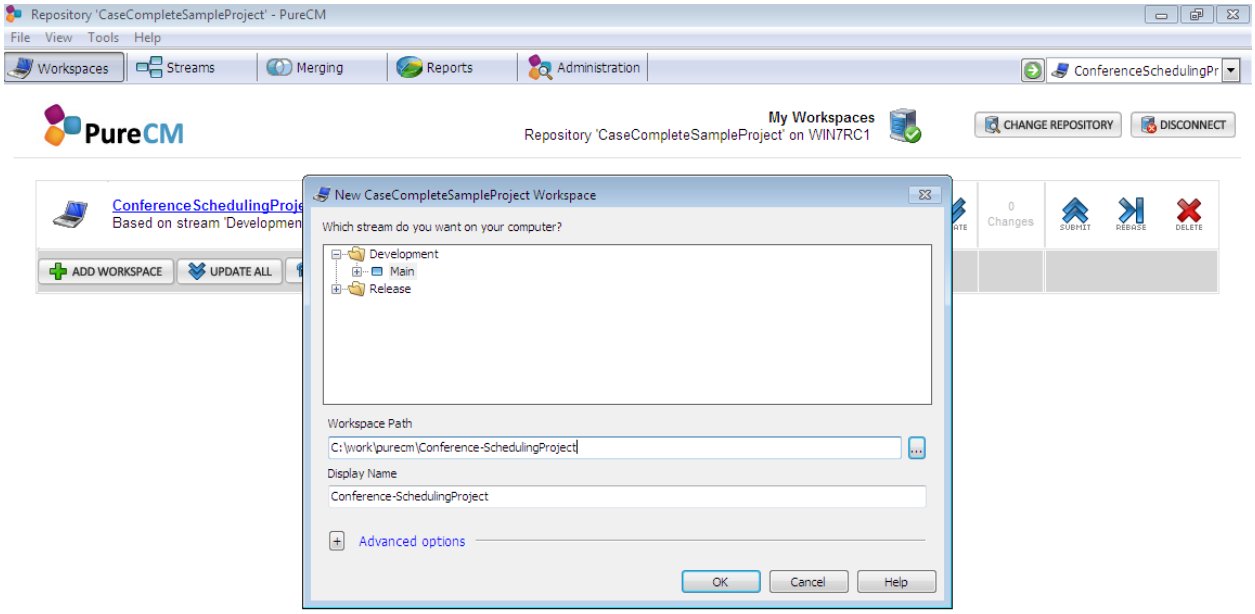


Click Submit. The project can now be checked out and modified by others who have workspaces for this stream.

Create a workspace for a shared project

Once you have created a shared project, other team members will want to access that shared project. Alternatively, if someone else has created a shared project, you will need to create a working copy of that project in order to review and update it.

Before you can start working on a shared project, you must create a workspace for that stream. Use the PureCM client to create a new workspace:



Once the workspace has been created, open the project from the newly created workspace by choosing File / Open from the CaseComplete menu. CaseComplete will open the project, and you can check out and modify it.

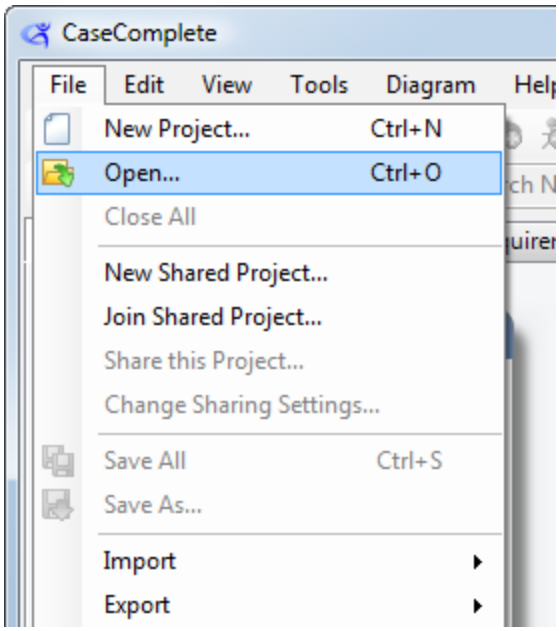
Note – once you have created a workspace to work on a shared project, you don't need to do these steps again – simply open the project from your workspace. CaseComplete will recognize it as a shared project and you'll be able to check out, update and check in project files.

Put an Existing Model under Version Control

If you have been working with earlier versions of CaseComplete, you probably have existing CaseComplete projects that you want to convert into sharable projects.

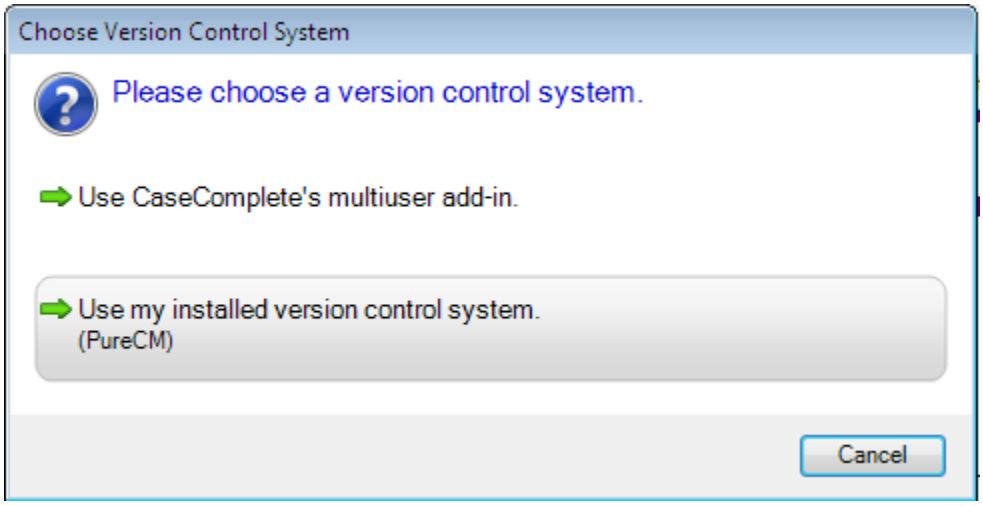
If you do not already have a workspace for the stream you want to use, create the workspace using the PureCM client.

Next, open the CaseComplete project using the Open option on the CaseComplete File menu:

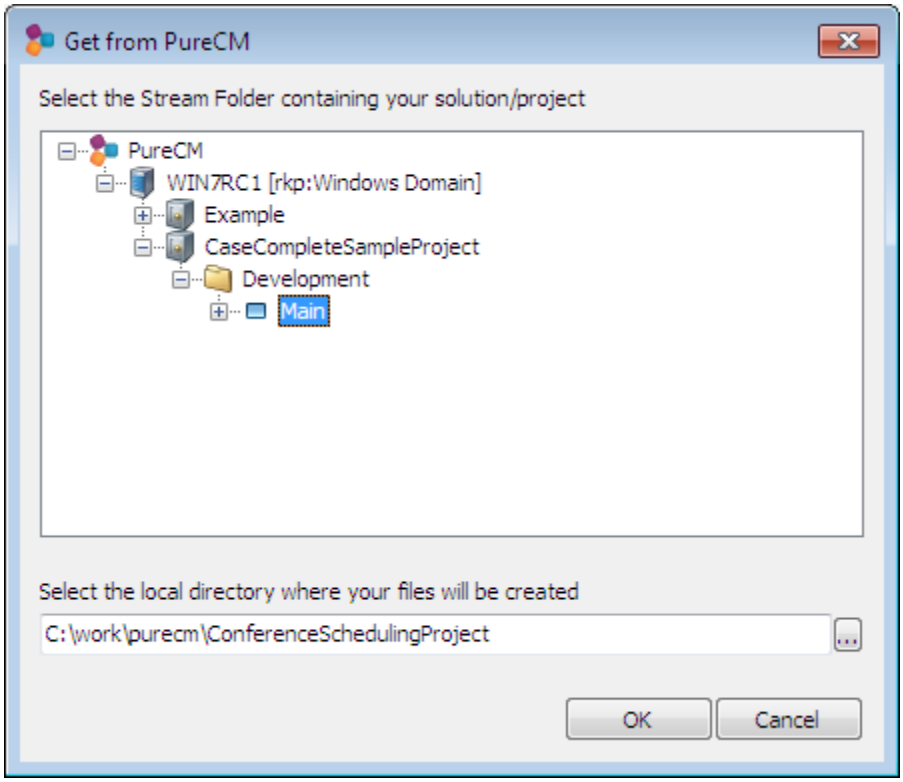


Once the model is open, select File / Save As. Choose your workspace folder as the target folder and click Save.

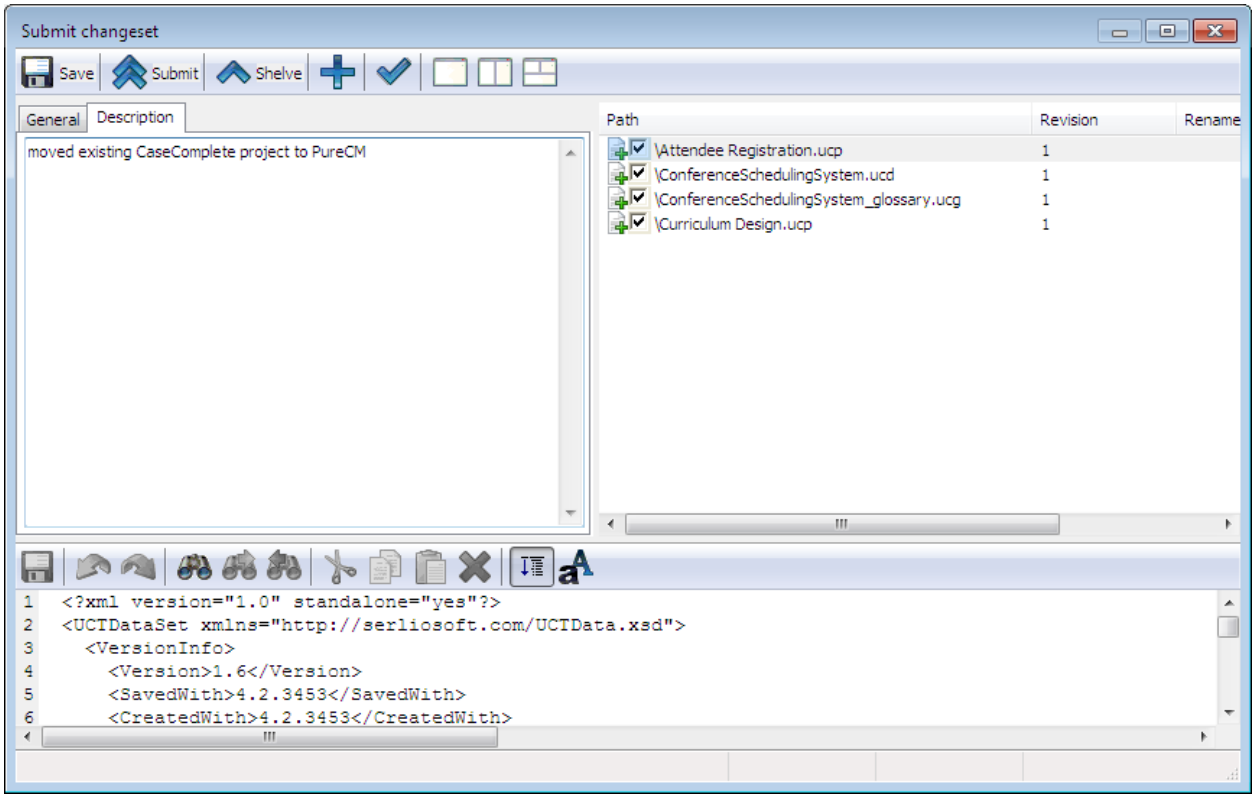
Next, select “Share This Project” from the CaseComplete File menu. CaseComplete will ask you which version control system to use. Select “Use my installed version control system”:



A PureCM dialog will display and allow you to confirm the stream and workspace associated with this project. Click Ok.



Next, PureCM will confirm your submission and allow you to annotate the checkin with comments:



Click Submit. The project can now be checked out and modified by others who have workspaces for this stream.

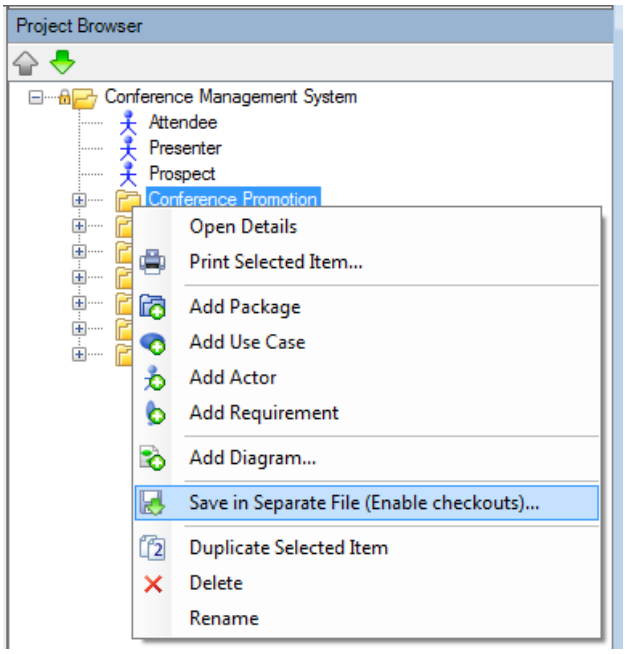
Note: Related documents referenced from a project created with an earlier version of CaseComplete are not automatically version controlled when you are starting with a project. To add those to version control, see "Add Related Documents To Version Control," below.

Save a Package as a Separate File

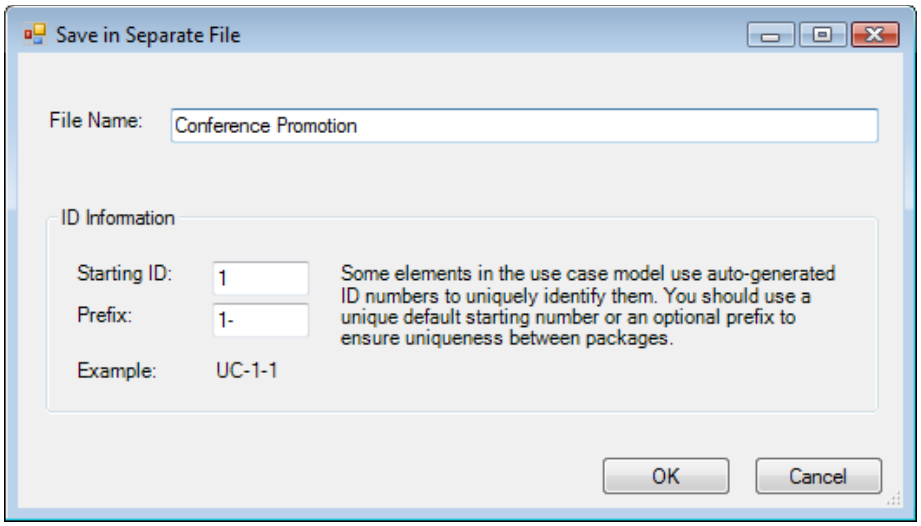
CaseComplete projects are created as a single file. As your project grows, you may want to break apart your project into packages so that those individual parts of your project can be checked out, modified and checked in independently.

Note: for more information on CaseComplete files, please refer to the section titled "CaseComplete Files" above.

To save a package in a shared project as a separate file, simply right-mouse click the package in the project explorer. Select the "Save in Separate File (Enable checkouts)…" menu item.



CaseComplete will prompt you for a file name. Simply enter in the name of the file:



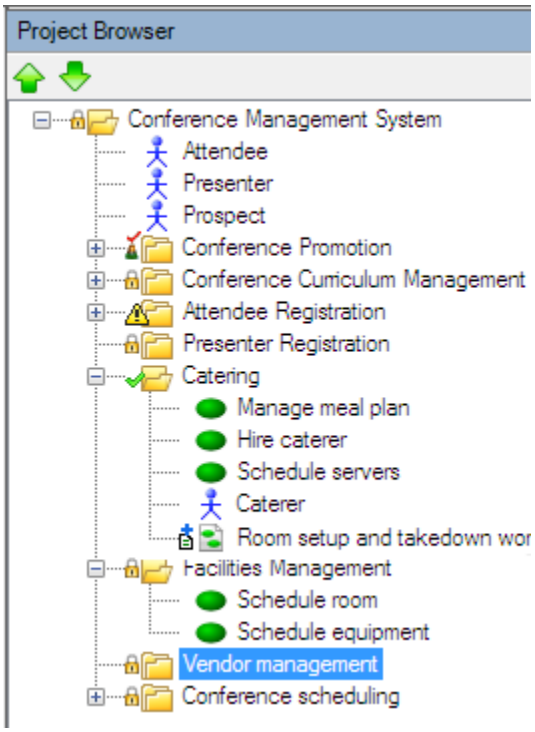
You can also specify a starting identification number and prefix for the elements that will reside in this package. This ensures that all identifiers are unique across your project (for example, you don't want to have two use cases with the id UC-1 in the same model, so the owning package can provide a unique qualifier for the package elements).

Click OK. CaseComplete will put the newly created file under version control. You can make modifications to the package, save and check in.

Note: Be sure to check in the new package and the owning package(s) at the same time, since other users won't be able to see the newly created package file without the reference to it that is created in the parent package(s).

See Checkout Status of Elements

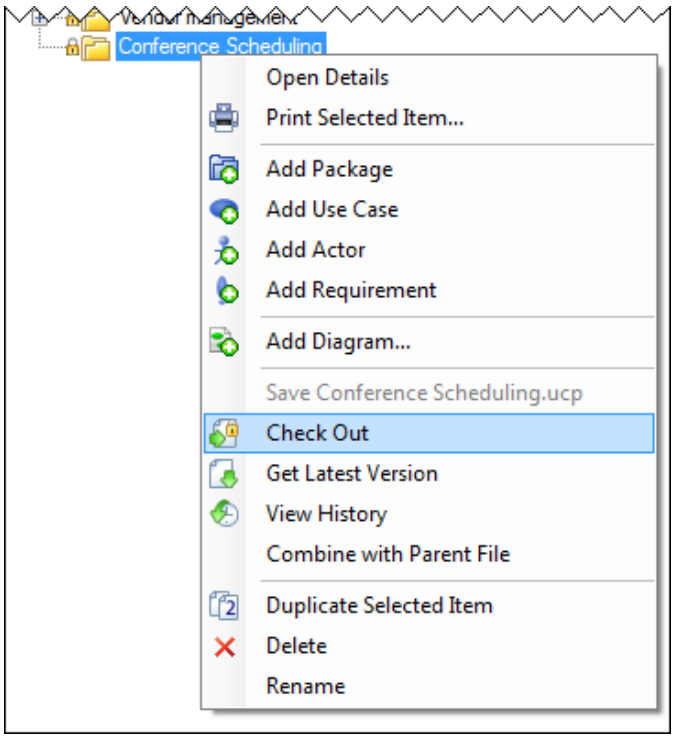
The CaseComplete project browser updates the icons displayed to indicate the status of packages and diagrams.



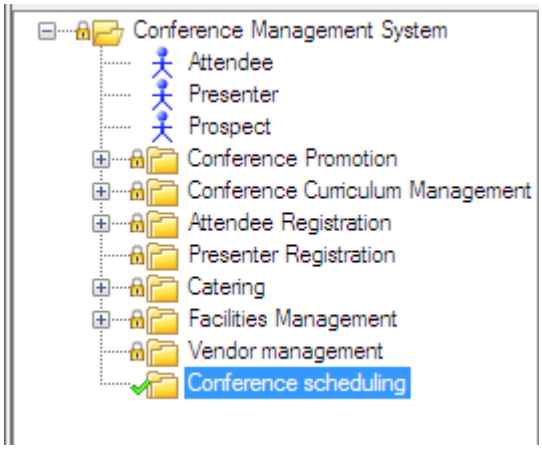
Symbol	Meaning	Description
	**Checked In**	When something is checked in and available for checkout, CaseComplete displays it with a lock. This indicates that it cannot be updated until it is checked out.
	**Checked Out**	When something is checked out to you, CaseComplete displays it with a green check. This indicates that it is checked out to you, and you can make changes. No one else can check it out while you have it checked out.
	**Locked**	When something is checked out to someone else, CaseComplete displays itwith a human figure with a red checkmark. This indicates that it cannot be checked out now.
	**New**	When you have added something new that hasn't been checked in yet, CaseComplete displays it adorned with a page with blue plus sign.
	**Out of date**	Someone may have changed something and checked it in since the last time you got the latest versions. CaseComplete indicates that by showing the icon adorned with a yellow "caution" triangle.

Check Out for Updating

Before you can make changes to a diagram or the contents of a package, it needs to be "checked out" so that it is locked from updates by other users. To check out a package or diagram, right-mouse click on it in the project browser and select Check Out.



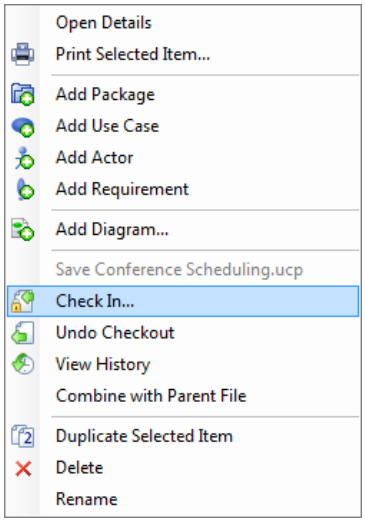
If the checkout is successful, its icon in the project explorer will be updated and will indicate its checked-out status with a green checkmark:



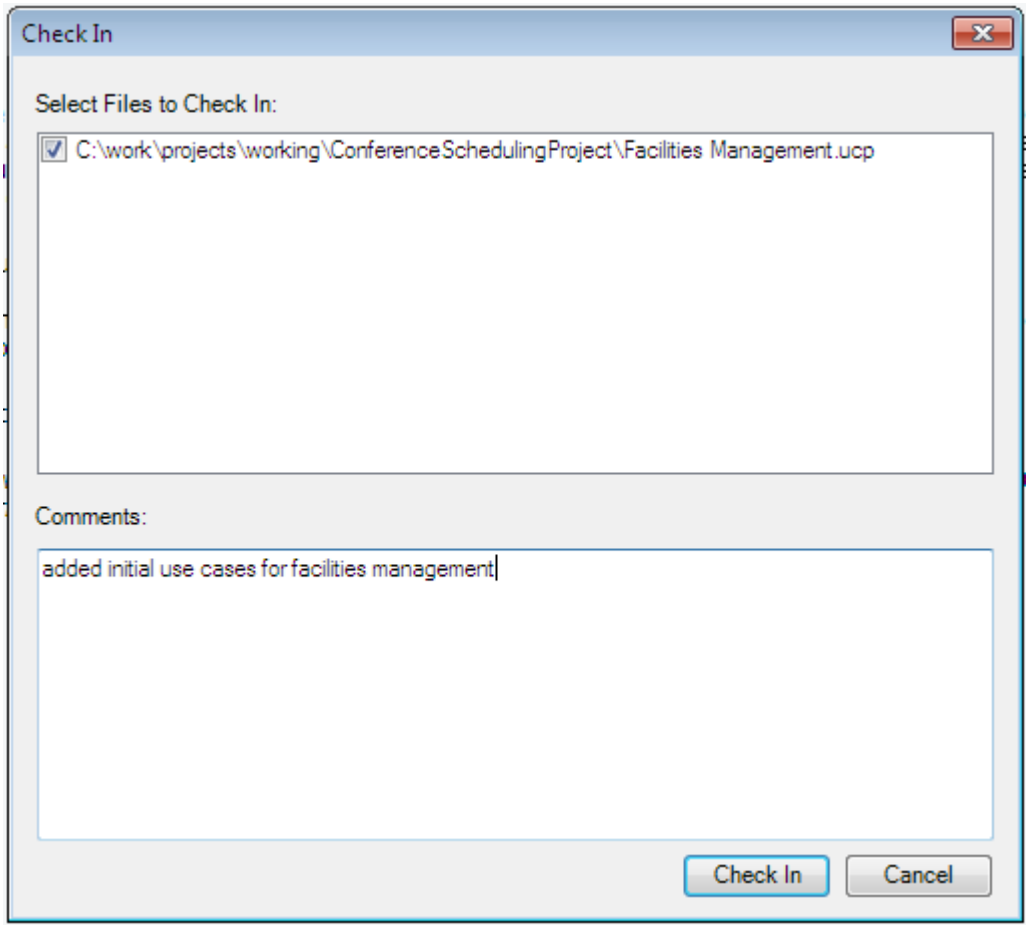
CaseComplete will display a message to let you know if a project element is already checked out by someone else. The message includes the user id of the person who has it checked out so you can coordinate changes with that person.

Check In Changes

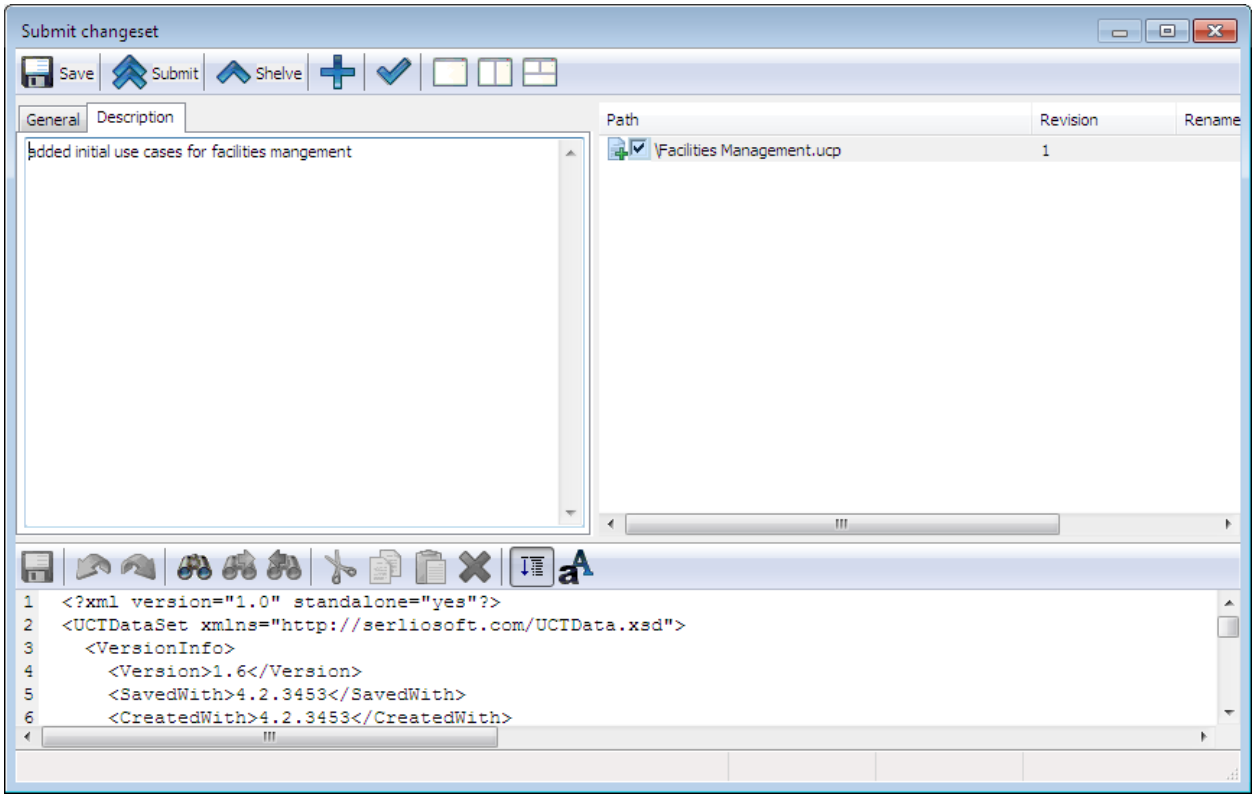
Once you have checked out something, you can then make changes to it. When you are done with the changes and are ready to commit your changes, right-mouse click the package and select “Check in” from the context menu:



When you select the Check In menu item, CaseComplete displays a dialog that displays what you are checking in, and allows you to enter comments to document the purpose of the change. Enter your comments and click the “Check In” button.



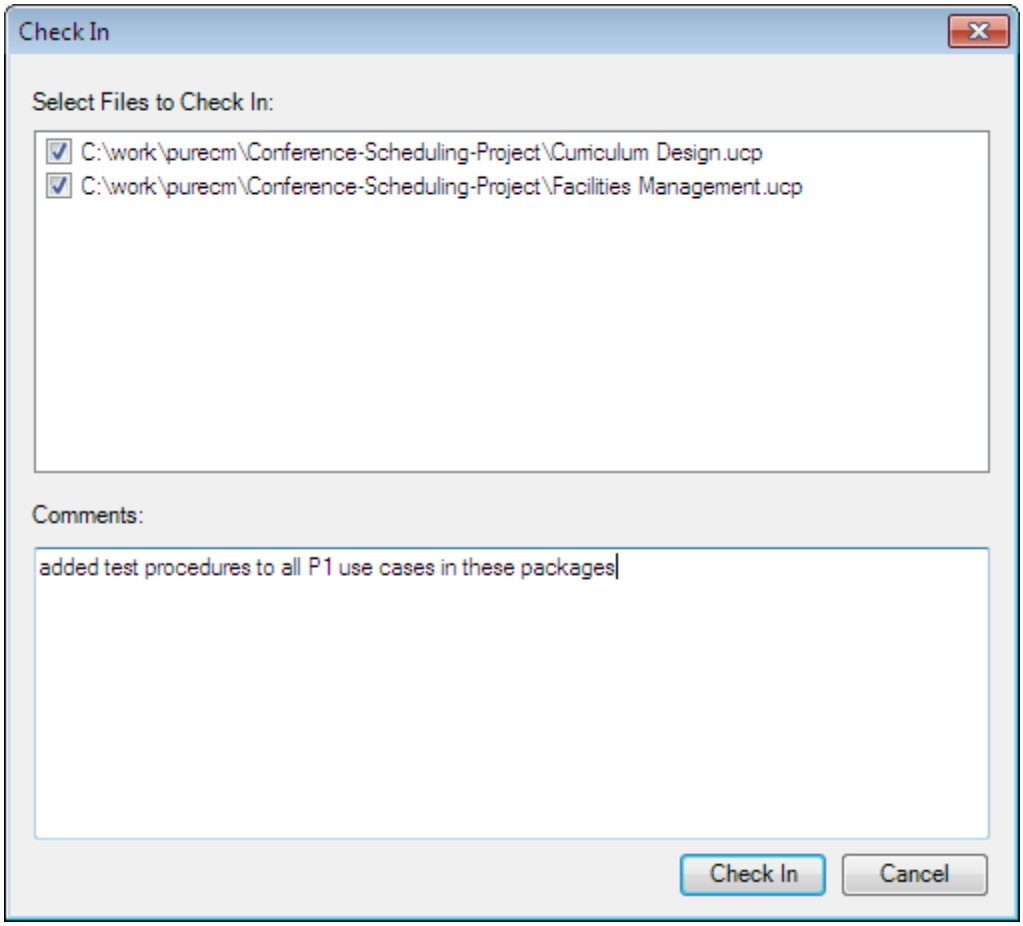
The PureCM Submit Changeset window will show your checkin. Click Submit:



After your changes have been checked in, the project explorer updates the element's icon indicating that it is checked in.

You can also check in several packages and/or diagrams at the same time. Click the Pending Checkins button on the toolbar:

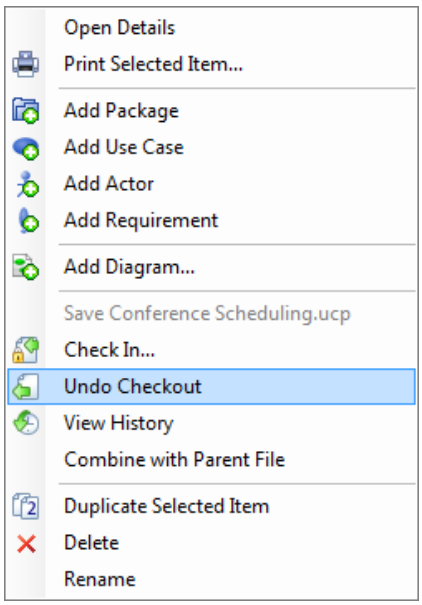
CaseComplete displays a dialog that displays what is checked out. Select the things that you want to check in, and enter comments to document the purpose of the change.



After your changes have been checked in, the project browser updates and displays the checked-in items with a lock icon.

Undo a Checkout

You may decide that the changes you are making are not necessary and want to abandon them. Undoing a checkout will release the checkout and refresh your working project with the most current version that has been checked in. To undo a checkout, right-mouse click the item that is checked out, and select Undo Checkout.

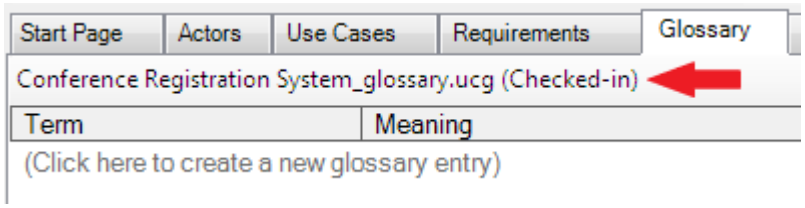


After the undo has been performed, your project will be updated with the most recent version and the icon for that item will be updated to reflect its current status.

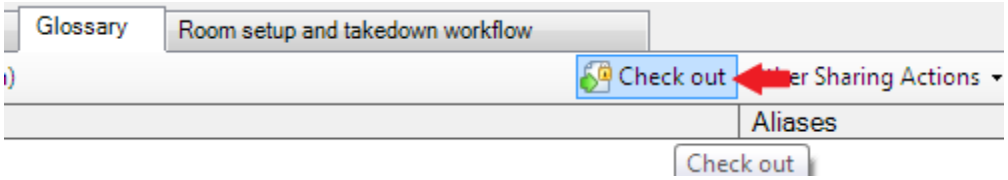
Working with the Glossary

The glossary for a CaseComplete project is stored as a separate file, and is controlled much like a package or diagram. The concepts of checking out, changing and checking in the glossary are just like other project elements. Because you work with a glossary in a different way than you might a package or diagram, though, the steps are a little different.

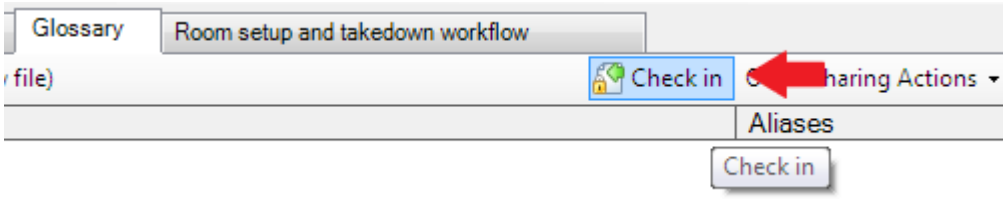
The status of the glossary is displayed at the top of the Glossary tab. It will indicate “checked in” if it is available for checkout, or will indicate to whom it is checked out otherwise:



To check out the glossary, go to the Glossary Tab and click the Check Out button:

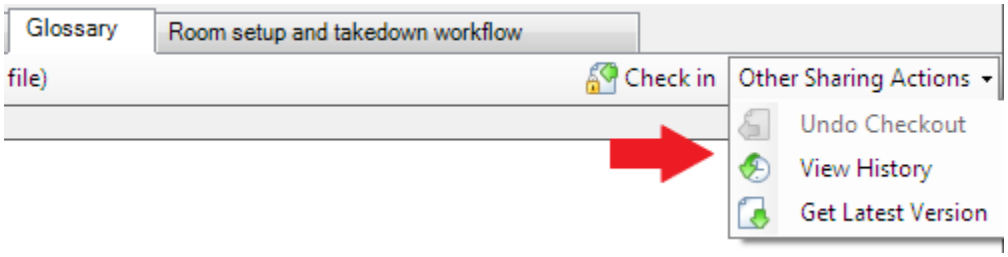


Once you have checked out the glossary, the Check Out button is replaced by a Check In button. To check in your changes, simply click the Check In button:



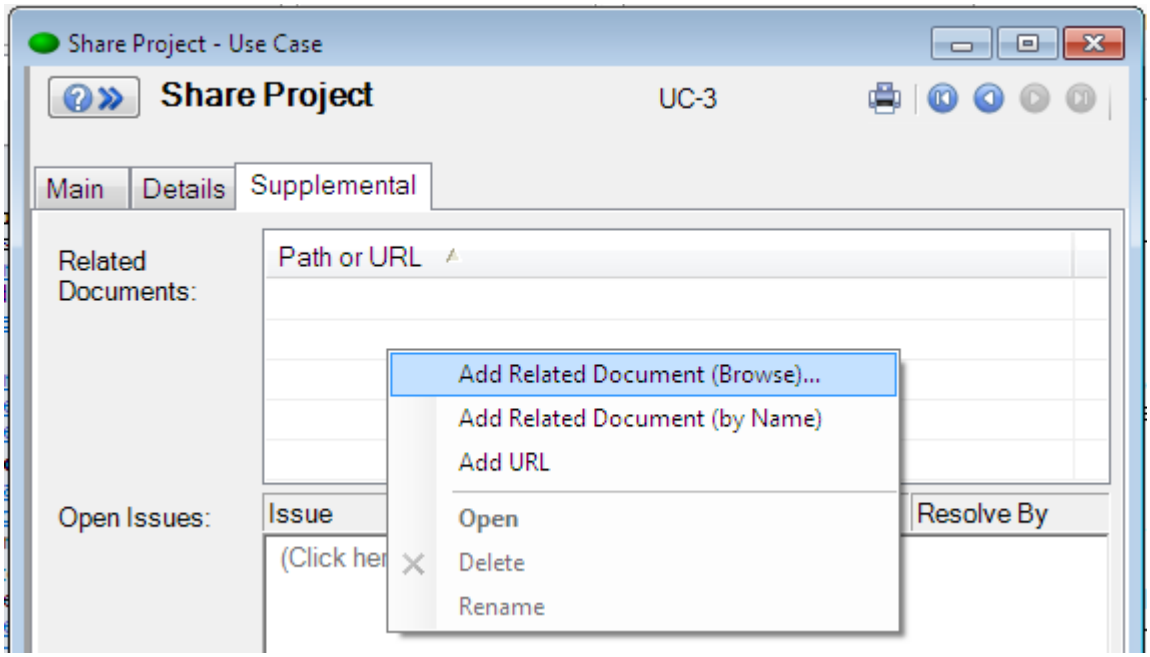
The PureCM Submit Changeset window will confirm your checkin.

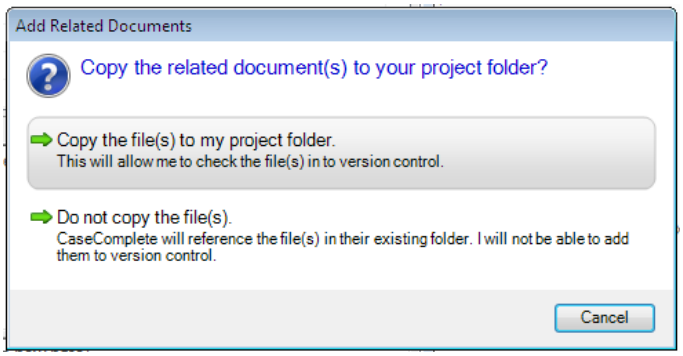
Options to undo your checkout, look at the glossary version history and get the latest version are available from the Other Sharing Actions dropdown in the Glossary tab:



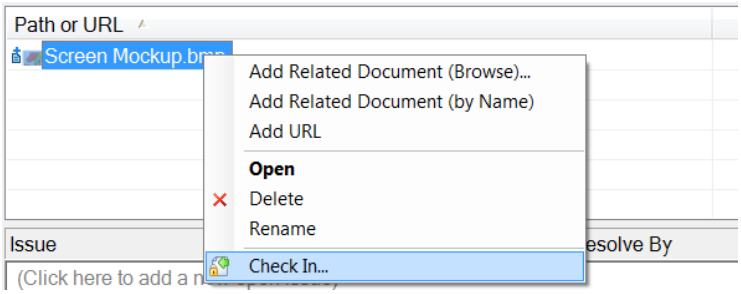
Add Related Documents to Version Control

Documents that are related to your use cases, actors and requirements can also be version controlled with CaseComplete. When you add a related document, CaseComplete will check to see where the document is located. If it is located outside of your project folder, CaseComplete will ask if you want the document to be moved into your project folder so it can be version controlled:

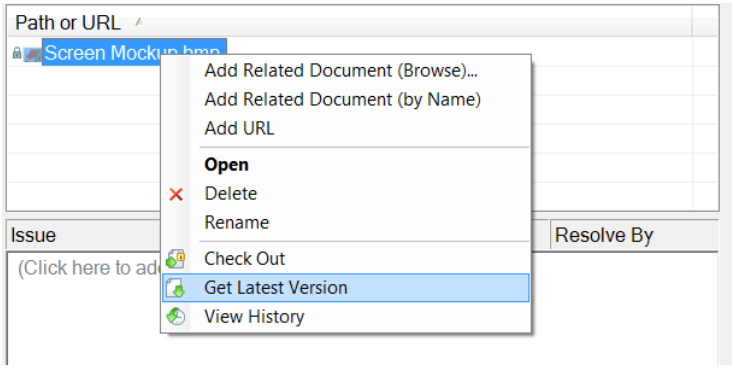




The document will display in the tab pane with the icon indicating the "new" checkin status. Check it in by right-mouse clicking on the document and selecting "Check In" from the context menu:



Once the document has been checked in, other version control commands are available from the same context menu:

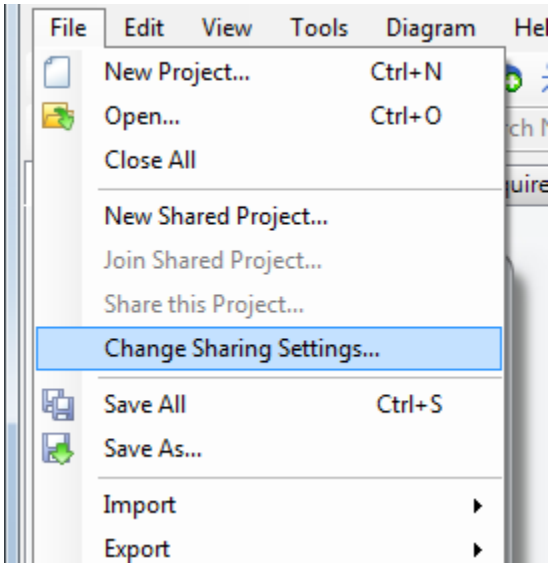


Update Version Control Settings for a Project

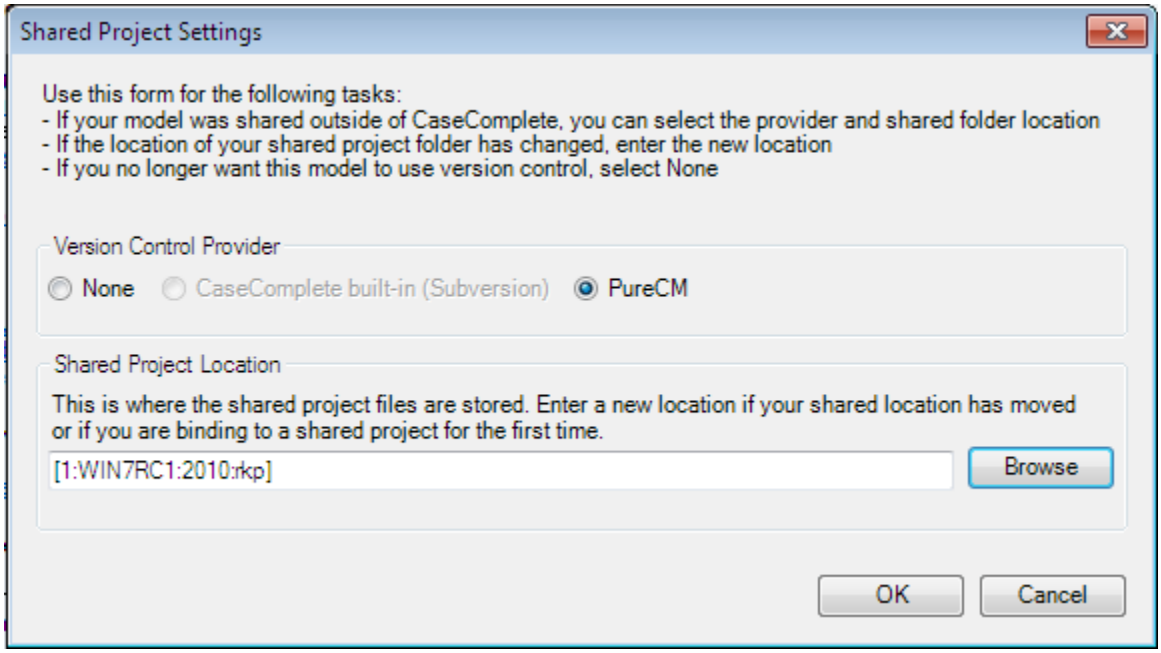
There may be occasions where you will need to change the version control bindings for a project:

- You no longer want to share the project
- You have a project under version control from an earlier version of CaseComplete

For this example, to change the version control settings, click “Change Sharing Settings…” from the File menu.



CaseComplete prompts you to choose the new setting for this project. To remove the project from version control, select “None.” Click Ok to confirm the changed setting.



Note: This action only changes CaseComplete’s project binding for version control. It will not remove the files from version control.