



Version Control Within UML Models Using Enterprise Architect

Enterprise Architect is an intuitive, flexible and powerful UML analysis and design tool for building robust and maintainable software.

This booklet explains the Version Control feature of Enterprise Architect.



Copyright © 1998-2010 Sparx Systems Pty Ltd

Version Control Within UML Models Using Enterprise Architect

© 1998-2010 Sparx Systems Pty Ltd

All rights reserved. No parts of this work may be reproduced in any form or by any means - graphic, electronic, or mechanical, including photocopying, recording, taping, or information storage and retrieval systems - without the written permission of the publisher.

Products that are referred to in this document may be either trademarks and/or registered trademarks of the respective owners. The publisher and the author make no claim to these trademarks.

While every precaution has been taken in the preparation of this document, the publisher and the author assume no responsibility for errors or omissions, or for damages resulting from the use of information contained in this document or from the use of programs and source code that may accompany it. In no event shall the publisher and the author be liable for any loss of profit or any other commercial damage caused or alleged to have been caused directly or indirectly by this document.

Printed: May 2010

Publisher

Sparx Systems

Managing Editor

Geoffrey Sparks

Technical Editors

Geoffrey Sparks

Howard Britten

Special thanks to:

All the people who have contributed suggestions, examples, bug reports and assistance in the development of Enterprise Architect. The task of developing and maintaining this tool has been greatly enhanced by their contribution.

Table of Contents

Foreword	1
Version Control	2
Version Control Basics	5
Apply Version Control To Models	6
Version Control & Team Deployment	7
Version Control Menu	8
Version Control Setup	9
Version Control Settings Dialog	9
Version Control Nested Packages	11
Version Control with SCC	11
Upgrade at Enterprise Architect 4.5	14
Version Control with CVS	15
CVS with Remote Repositories	15
CVS with Local Repositories	19
Version Control with Subversion	22
Set up Subversion	22
Create a new Repository Sub-tree	23
Create a Local Working Copy	24
Subversion Under WINE-Crossover	24
Version Control Configuration	26
TortoiseSVN	28
Version Control with TFS	28
Connect an Enterprise Architect Model to Version Control using TFS	29
Use Version Control	33
Package Version Control Menu	33
Configure Controlled Package	36
Use Existing Configuration	37
Validate Package Configurations	37
Check In and Check Out Packages	38
Include Other Users' Packages	40
Apply Version Control To Branches	41
Export Controlled Model Branch	42
Import Controlled Model Branch	42
Review Package History	44
Refresh View of Shared Project	44
Resynchronize the Status of Version Controlled Packages	45
Offline Version Control	46
Index	48

Foreword

This user guide provides an introduction to the Version Control feature of Enterprise Architect.

Version Control



Enterprise Architect supports version control of packages and their component sub-packages to a central version control repository. You can place any individual packages, View nodes or model root nodes under version control.

Features

Version Control provides two key facilities:

- Coordinating sharing of packages between users
- Saving a history of changes to Enterprise Architect packages, including the ability to retrieve previous versions.

System Requirements

Version controlled packages are packages that have been configured for use with version control software. To use version control in Enterprise Architect, a third-party source-code control application is required that controls access to and stores revisions of the controlled packages. Enterprise Architect supports the following version control applications:

- Subversion, which is available from <http://www.subversion.tigris.org/>
- CVS, which is available from <http://www.march-hare.com/cvspro/>
- Microsoft Team Foundation Server
- SCC-compatible products; all version control products that provide a client that complies with the Microsoft Common Source Code Control standard, version 1.1 or higher.

The following products are SCC-compatible and are known to successfully integrate with Enterprise Architect:

- Accurev	Tested by Sparx	
- Borland Star Teams		Users report success
- ClearCase		Users report success
- MS Visual Source Safe	Tested by Sparx	
- MS TFS-SCC	Tested by Sparx	
- MKS Source Integrity	Tested by Sparx	
- Perforce	Tested by Sparx	
- Serena Dimensions		Users report success
- Serena Change Manager		Users report success
- Snapshot CM	Tested by Sparx	
- SourceGear Vault	Tested by Sparx	
- Source Offsite	Tested by Sparx	

Products that do not appear in the list should still integrate successfully with Enterprise Architect, if there is a client available for that product that complies with the MS SCC API specification.

Set-Up

Before using Enterprise Architect's version control facility, your version control software must be installed on each machine on which it is intended to be used.

Typically there are:

- A server component that manages a version control repository, and
- Client components on the workstations that Enterprise Architect uses to communicate with the server.

A version control client must be installed on every machine where you run Enterprise Architect and want to access your version control system. Once the version control software has been installed and configured, you must define a Version Control Configuration within Enterprise Architect, to use your installed version control product.

Note:

Sparx Systems strongly urge you not to manipulate version controlled package files outside of Enterprise Architect. It is possible to leave the package files in a state that Enterprise Architect cannot recognize.

Usage




There are four basic ways in which the version control facility might be used:


Use	Description
Single Shared model	Users share an Enterprise Architect model, stored in a central .EAP file or DBMS repository. This configuration enables you to view changes to other users' packages without explicitly having to check them out, but by simply refreshing your view of the model. <ul style="list-style-type: none"> • Version control regulates access to packages, and maintains package revision history.
Multiple Private models	An Enterprise Architect model is created by a single user who configures it for version control. The model file is then distributed to other users, with each user storing their own private copy of the model. <ul style="list-style-type: none"> • Users update their model's packages through version control • Version control regulates access to packages, and maintains package revision history • Other users' new packages are retrieved using the Get Package menu option.
Shared packages	Individual users create separate Enterprise Architect models but share one or more packages. <ul style="list-style-type: none"> • Users share packages through version control.
Standard packages	A company might have a standard set of packages which are broadly shared (on a read-only basis). <ul style="list-style-type: none"> • Individual users retrieve packages with the Get Package menu option.

For a discussion of how each of these arrangements might be used, see the Version Control white paper: <http://www.sparxsystems.com/resources/whitepapers/index.html>.

Version Control Indicators

Packages under version control are identified in the **Project Browser** by icons that indicate the current status of the package.

Icon	Indicates that...
	This package is controlled and is represented by an XMI file on disk. Version control either is not being used or is not available. You can edit the package. (See the <i>Controlled Packages</i> topic in <i>UML Model Management</i> .)
	This package is version controlled and checked out ³⁸ to you, therefore you can edit the package.
	This package is version controlled and not checked out to you, therefore you cannot edit the package (unless you check the package out).

Icon	Indicates that...
	This package is version controlled, but you checked it out whilst not connected to the version control server. You can edit the package but there could be version conflicts when you check the package in again.

For example, below, *CVS00* and *CVSPackage* are configured for version control. *CVS00* is checked out to you, and *CVSPackage* is not.



See Also

- [Version Control Basics](#) ⁵
- [Apply Version Control To Models](#) ⁶
- [Version Control and Team Deployment](#) ⁷
- [Version Control Menu](#) ⁸
- [Version Control Setup](#) ⁹
- [Use Version Control!](#) ³³
- [Offline Version Control!](#) ⁴⁶

1 Version Control Basics

The Lock-Modify-Unlock Solution

Many version control systems use a lock-modify-unlock model to address the problem of different authors in a shared source overwriting each other's work. In this model, the version control repository allows only one person to change a file at a time, and access is managed using locks. Harry must lock a file before he can begin making changes to it. If Harry has locked a file, Sally cannot also lock it, and therefore cannot make any changes to that file. All she can do is read the file, and wait for Harry to finish his changes and release the lock. After Harry unlocks the file, Sally can take her turn in locking and editing the file.

The Copy-Modify-Merge Solution

Subversion, CVS and a number of other version control systems use a copy-modify-merge model as an alternative to locking. In this model, each user's client contacts the project repository and creates a personal working copy—a local reflection of the repository's files and directories. Users then work simultaneously and independently, modifying their private copies. In due course, the private copies are merged together into a new, final version. The version control system often assists with the merging, but ultimately a person is responsible for making it happen correctly.

When Locking is Necessary

While the lock-modify-unlock model is generally considered a hindrance to collaboration, there are still times when locking is necessary.

The copy-modify-merge model is based on the assumption that files are contextually merge-able: that is, the files in the repository are line-based text files (such as program source code). But for files with binary formats, such as artwork or sound, it is often impossible to merge conflicting changes. In these situations, it really is necessary for users to take strict turns in changing the file. Without serialized access, somebody ends up wasting time on changes that are ultimately discarded.

2 Apply Version Control To Models

All Enterprise Architect models are stored in databases - even the .EAP file is a database. In simple, version control terms, the model is a single entity of binary data. It is not practical to apply version control to the database as a whole. Being binary data, it would require the use of the [lock-modify-unlock model](#)^[5] of version control, which would mean that only a single user at a time could work on any given (version controlled) model.

To overcome this limitation, Enterprise Architect exports discreet units of the model - the packages - as XML package files, and it is these XML files, not the .EAP file, that are placed under version control. The XML file format used by Enterprise Architect dictates that they too be treated as binary files (therefore it is not possible to merge the XML files either); however, by splitting the model into much smaller parts, this approach enables many users to work on separate parts of the model simultaneously.

When a user [checks-out](#)^[38] a package, Enterprise Architect sends a command to the version control system to check-out the equivalent XML file. The version control system then puts the latest revision of the file into the user's working copy directory, overwriting any previous revision of the file in that directory. Enterprise Architect then imports the package file into the model, updating the contents of the existing package in the model.

When [checking-in](#)^[38], Enterprise Architect exports the package as an XML file, overwriting the existing local working copy of the file. The new file is then checked-in to the version control system.

Nested Version Controlled Packages

Nested version controlled packages result in much smaller XML files being exported for parent packages, as the parent packages' XML files do not contain any content for the version controlled child packages.

[Version Control of nested packages](#)^[11] together with a model structure having small individual packages also provides greater scope for multiple users to work concurrently, as individual users are locking much smaller parts of the model.

Notes:

- **Do not place your .EAP files under version control**, as this creates problems for you.
- Most version control systems mark their controlled files as read only, unless they are specifically checked-out to you.
- The .EAP file is an MS Jet database, and Enterprise Architect **must** be able to open this file for read/write access when you load your model. (Enterprise Architect displays an error message and fails to load the model if it is read-only.)

3 Version Control & Team Deployment

Team deployment and the use of version control is discussed in two Sparx Systems white papers, available on the Sparx Systems web site:

- http://www.sparxsystems.com/WhitePapers/Version_Control.pdf
- http://www.sparxsystems.com/downloads/whitepapers/EA_Deployment.pdf

A brief summary of the process is provided below:

1. Install your version control product.
2. Create a version control repository.
3. Create a version control project to be used with your Enterprise Architect project, and check-out a working copy of the project into a local folder. (You must do this for every team member that is accessing the version controlled packages, whether you are using a single shared model or each team member stores his own private copy of the model.)
4. Within Enterprise Architect, [define a version control configuration](#)^[9] to provide access to the working copy files. Again, each user must do this on their own workstation, as the details are stored within the Windows registry.
5. [Configure packages](#)^[36] within the Enterprise Architect model for version control. That is, apply version control to individual packages.
6. [Check-out and check-in packages](#)^[38] as required.

Note:

The name of the version control configuration must be the same across all machines. That is, all version control access to a given Enterprise Architect package must be through version control configurations with the same name, across all models and all users. (It is possible to use multiple version control configurations within the same model, so different packages can still use different version control configurations within the same model, as long as any given package is always accessed via the same version control configuration.)

The easiest way to perform step 4, (throughout the team), is to have one user set up version control on the model and then share that model with the rest of the team.

- In Shared Model deployment, all users connect to a single instance of the model database, so the model is shared automatically.
- In Private Model deployment, it is easiest to distribute copies of the original model (after version control has been set up) to all other members of the team.

Whenever you open a model ([Private or Shared](#)^[3]) that uses a version control configuration that is not yet defined on your workstation, Enterprise Architect prompts you to complete the definition for that configuration. This typically means specifying the local working copy directory and maybe choosing the version control project associated with this Enterprise Architect project.

Once this has been done, the version controlled packages that already exist in the model are ready for use.

Version Control Branching

Currently, Enterprise Architect does not support Version Control Branching. Work-arounds to achieve similar results might be possible for certain version-control products; contact Sparx Support for advice:

- Registered users - http://www.sparxsystems.com/registered/reg_support.html
- Trial users - support@sparxsystems.com.

4 Version Control Menu

You access the **Version Control** menu through the **Project | Version Control** menu option. It provides the following options:

Menu Option & Function Keys	Use to
Configure Current Package [Ctrl]+[Alt]+[P]	Display the Package Control Options ^[36] dialog, which enables you to specify whether this package (and its children) is version controlled, and which version control configuration applies.
Version Control Settings	Display the Version Control Settings dialog ^[9] .
Validate Package Configurations	Test the validity ^[37] of the version control settings associated with each version controlled package within your current model.
Re-Synch Statuses of All Packages	Resynchronize the version control status of packages ^[45] as recorded in your Enterprise Architect project when they are out of synchronization with the version control status reported by your version control provider. The function acts on all version controlled packages within the Enterprise Architect project, updating the values recorded in the project to match the values reported by the version control provider, without performing any XML import or export.
Work Offline	Work independently of the version control server ^[46] , if it is unavailable to you.

5 Version Control Setup

Before using Enterprise Architect's version control facility, your version control product must be installed on each machine where it is intended to be used. Version Control products supported by Enterprise Architect include MS Team Foundation Server, Subversion, CVS or any other version control product that provides an MS SCC-compliant interface.

- Subversion is available from <http://www.subversion.tigris.org/>
- CVS is available from <http://www.wincvs.org/>.

Note:

If you are using the Corporate, Business and Software Engineering, System Engineering or Ultimate editions of Enterprise Architect with security enabled, you must also set up permissions to configure and use version control. See the *List of Available Permissions* topic in *User Security in UML Models* for further information.

Typically there should be:

- A server component that manages a version control repository
- Client components on the workstations that Enterprise Architect uses to communicate with the server.

A version control client must be installed on every machine where you run Enterprise Architect and want to access your version control system. Once the version control software has been installed and configured, to use your installed version control product you must define a Version Control Configuration within Enterprise Architect.

Version control can be assigned to individual packages, view nodes or root nodes in Enterprise Architect. Each package can only be linked to one Version Control Configuration at a time, although it is possible to connect multiple control configurations for each model. You can use the **Version Control Settings** dialog to set up a connection to your version control application.

To set the Version Control Configuration, select the **Project | Version Control | Version Control Settings** menu option, and see the [Version Control Settings Dialog](#)⁹ topic.

See Also

- [Version Control with SCC](#)¹¹
- [Version Control with CVS](#)¹⁵
- [Version Control with Subversion](#)²²
- [Version Control with TFS](#)²⁸

5.1 Version Control Settings Dialog

The **Version Control Settings** dialog enables you to specify the information required to create a Version Control Configuration, which can then be used to establish a connection to a version control provider. Enterprise Architect supports version control through MS Team Foundation Server, Subversion, CVS or any SCC-compliant version control product.

It is possible to use multiple version control configurations in the same Enterprise Architect model. It is also possible to use the same version control configuration across different models, to facilitate sharing 'standard' packages between those models, through the version control system.

Note:

In the Corporate, Business and Software Engineering, System Engineering and Ultimate editions of Enterprise Architect, if security is enabled you must have **Configure Version Control** permission to set up version control options for the current model. See *User Security in UML Models*.

Setting Up Version Control

When you display the **Version Control Settings** dialog for the first time in any given model, it appears as shown below:

Model Settings

This model is private (for Shared models, it is best to disable this check box)

Save nested version controlled packages to stubs only (recommended)

Configuration Details:

Unique ID:

Type: SCC CVS Subversion TFS

Defined Configurations:

Unique ID	Type	Files	Location
-----------	------	-------	----------

To begin defining a new version control configuration, follow the steps below:

1. Click on the **New** button.
2. In the **Unique ID** field, type a suitable name.
3. Against the **Type** field, click on the radio button for the version control product to connect to.

At this point, the middle section of the dialog changes to display a collection of fields relating to the type of Version Control Configuration you are defining. Go to the relevant topic below:

- [Version Control with SCC](#) ^[11]
- [Version Control with CVS](#) ^[15]
- [Version Control with Subversion](#) ^[22]
- [Version Control with TFS](#) ^[28]

To import a previously defined configuration for use in the current model, follow the steps below:

1. Click on the **New** button.
2. In the **Unique ID** field, click on the drop-down arrow and select one of the previously defined version control configurations.
3. Click on the **Save** button to save the selected version control configuration in this model.

See Also

- [Version Control Nested Packages](#)^[11]

5.1.1 Version Control Nested Packages

In releases of Enterprise Architect later than version 4.5, when you save a package to the version control system only stub information is exported for any nested packages. This ensures that information in a nested package is not inadvertently over-written by a top level package.

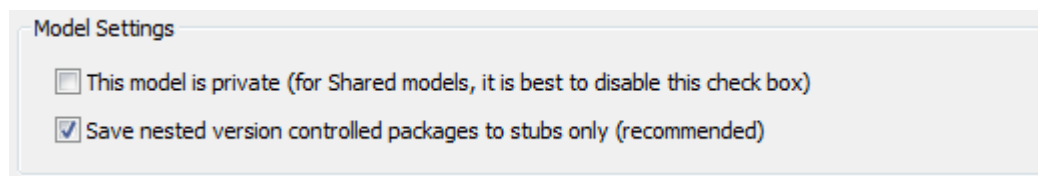
When checking out a package, Enterprise Architect does not modify or delete nested packages; only the top level package is modified.

As a consequence of this behavior, if you check out or get a version controlled package with nested packages not already in your model, you see stubs in the model for the nested packages only. If you select the **Get All Latest** option from the version control menu, Enterprise Architect populates these new stubs from the version control system.

Using the above technique you can populate a large and complex model from only the root packages, using **Get All Latest** to recursively iterate through the attached and nested packages.

This is a powerful and efficient means of managing your project and simplifies handling very large models, even in a distributed environment.

It is recommended you do not mix versions of Enterprise Architect later than version 4.5 with earlier versions when sharing a version controlled model. If this is necessary it is best to go to the [Version Control Settings](#)^[9] dialog and deselect the **Save nested version controlled packages to stubs only** checkbox, setting Enterprise Architect to the pre-version 4.5 behavior (for the current model only).



5.2 Version Control with SCC

To set up an SCC version control configuration, you must:

- Set up the source code control provider with SCC, and
- Connect the Enterprise Architect model to version control with SCC.

See also, the topic on version control with SCC when [Upgrading at Enterprise Architect 4.5](#)^[14].

Set Up the Source Code Control Provider with SCC

To set up the third-party source code control provider, see the documentation provided with that application. A repository must be set up using the SCC provider and access to that repository must be available to all intended users.

Connect an Enterprise Architect Model to Version Control with SCC

To connect an Enterprise Architect model to version control, follow the steps below:

1. Open or create the Enterprise Architect model to place under version control.
2. Select the **Project | Version Control | Version Control Settings** menu option. The **Version Control Settings** dialog displays.

Model Settings

This model is private (for Shared models, it is best to disable this check box)

Save nested version controlled packages to stubs only (recommended)

Configuration Details:

Unique ID:

Type: SCC CVS Subversion TFS

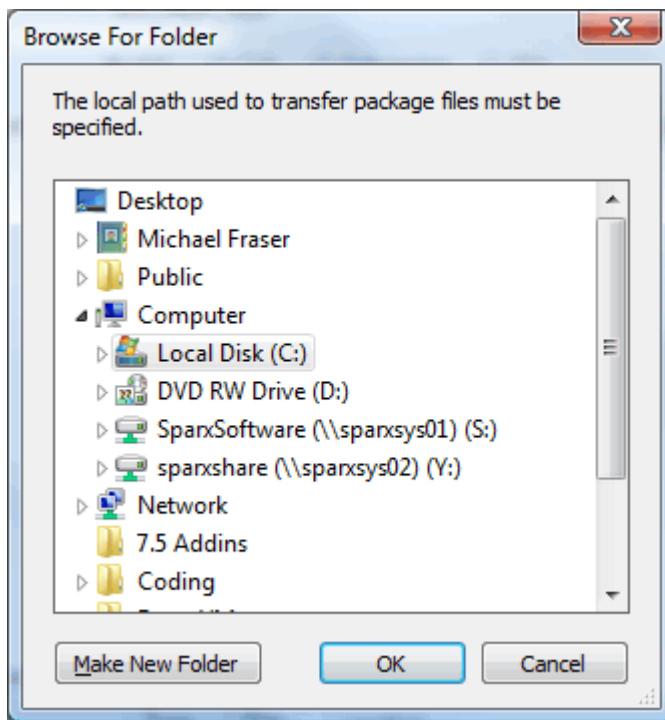
New Save Delete

Defined Configurations:

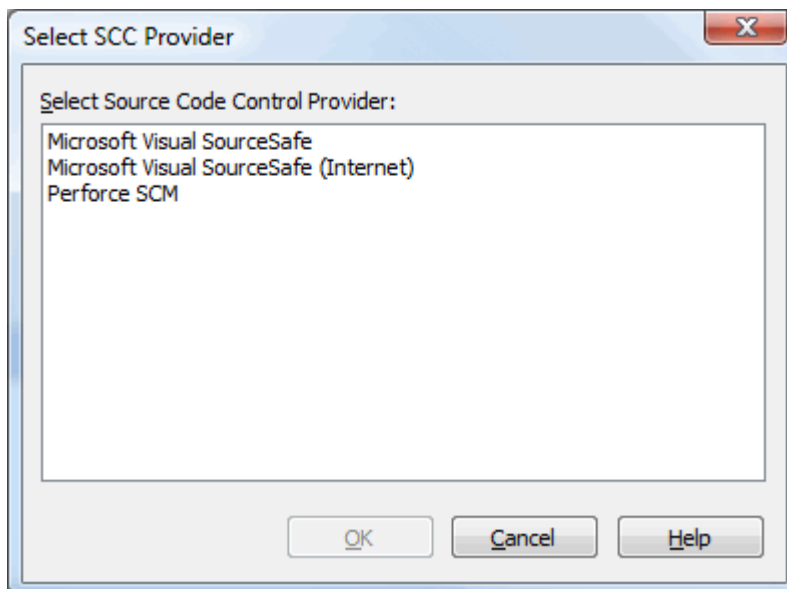
Unique ID	Type	Files	Location
-----------	------	-------	----------

Close Help

3. Click on the **New** button.
4. In the **Unique ID** field, type a suitable name. Click on the **SCC** radio button.
5. To the right of the **Local Project** path field, click on the **Select Path...** button. The **Browse for Folder** dialog displays.



6. Locate and click on the local folder in which to keep local working copies of the XML files to be stored in the Version Control repository.
7. Click on the **OK** button. The **Select SCC Provider** dialog displays.

**Note:**

All users of the shared database must specify the same SCC provider.

8. Click on an SCC provider, and click on the **OK** button to return to the **Version Control Settings** dialog.
9. Click on the **Save** button to save the configuration you have defined.

The SCC provider is likely to prompt you for various details including the name of the project to connect to, and perhaps the user name to use when you log in.

10. The new configuration is added to the list in the **Defined Configurations** panel.

Note:

A new entry is also created in the Local Paths list, with the same ID as the new version control configuration. The **Local Path** entry records the Local Project path, for use in subsequent path substitutions. See the *Local Paths Dialog* topic in *Code Engineering Using UML Models*.

11. When you have finished defining your version control configurations, click on the **Close** button. For further information on the fields on the **Version Control Settings** dialog, see the following table.

Field	Use to
This model is private	Specify whether all users connect to a single shared copy of the model (for example, a DBMS) or each user connects to their own private copy of the model. When unselected (for shared models), the option disables the File History - Retrieve functionality when the selected package is checked out by another user. This prevents modifications that might have been made by the other user from being discarded through importing a prior revision from version control.
Save nested version controlled packages to stubs only	Set nested version controlled packages to stubs or fully expanded trees. Defaults to selected. For a full explanation of this option, see Version Control Nested Packages ¹¹ .
Unique ID	Specify a configuration name that readily distinguishes this configuration from other configurations. The unique ID is displayed as a selection in the list of Version Control configurations a package can connect to. You can also click on the drop-down arrow and select a previous version control configuration, providing the configuration is not in the current model.
Local Project Path	Specify the folder in which the XML files representing the packages are stored. This folder should already exist before it is specified here. Every PC using version control should have its own local SCC project folder, and this should not be a shared network folder. Particularly bear this in mind if you are creating a .EAP file that is to be shared (such as a SQL database).
Current User	Read only. Shows your user name as the user currently logged into the SCC provider.
SCC Provider	Read only. Shows the name of the provider specified in the database.
SCC Project	Read only. Shows the project selected during the initial setup of the connection to the SCC provider.

Note:

Sparx Systems strongly urge you not to manipulate version controlled package files outside of Enterprise Architect. It is possible to leave the package files in a state that Enterprise Architect cannot recognize.

5.2.1 Upgrade at Enterprise Architect 4.5

When a version-controlled project created under a release of Enterprise Architect earlier than 4.5 is opened in Enterprise Architect release 4.5 or later, you must identify the SCC connection with a new unique ID. You can assign a name to the existing SCC configuration or associate the project with a configuration that has previously been assigned a unique ID.

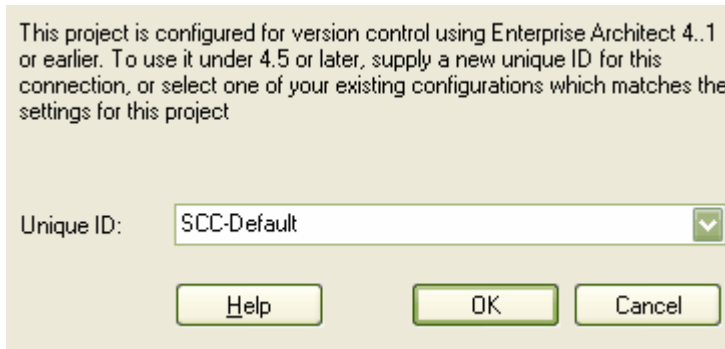
By having a unique ID for Version Control Configurations, you can assign a configuration quickly and efficiently using configurations that have been created previously for other version controlled repositories. This enables you to configure the many packages to use an existing version control repository; this can apply to packages created for more than just one model enabling a great deal of flexibility.

To upgrade an existing SCC version control project created before release 4.5, in Enterprise Architect release 4.5 or later, follow the steps below.

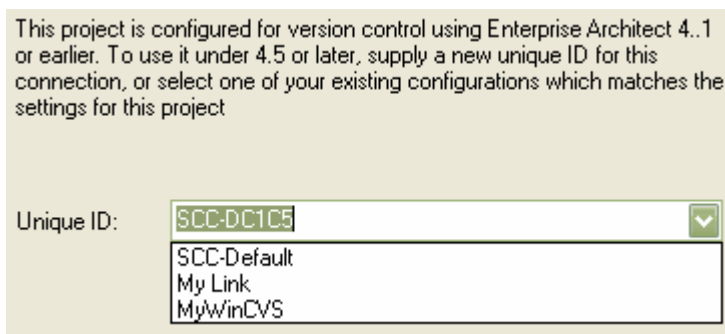
1. Open the project that has an SCC Version Control Configuration created in Enterprise Architect earlier

than version 4.5.

2. The **Select or Create Unique ID for Version Control** dialog prompts you to create an ID for an existing configuration or to choose a previously created one from the **Unique ID** drop-down list.
3. The existing SCC configuration is the initial value, represented by **SCC-XXXXX**; this number is not especially meaningful, therefore it is recommended that the configuration be given a meaningful name.



4. You can associate the version controlled package with a previously-defined configuration by selecting an existing configuration from the **Unique ID** drop-down list (if one exists).



5. After you have assigned the unique ID, click on the **OK** button to load the model.

5.3 Version Control with CVS

CVS is used to manage files and directories and is an open source, version control system. In order to use CVS version control with Enterprise Architect, you must install version control software on your local machine. Also, you must create a working directory (using your version control software) before you can configure Enterprise Architect. You can have as many working directories as you like on your local machine.

You must also connect to a repository, which can be either a [remote repository](#)^[15] or [local](#)^[19] to your machine. If your repository is local, it must be created with your version control software.

Each working folder you create contains information on connection to a repository. This connection information includes the path to the local or remote repository, the user name and password in order to make a connection.

Note:

To see a video demonstration of setting up a CVS repository for version control, go to http://www.sparxsystems.com.au/resources/demos/settingupCVS/CVS_Final_1.htm.

5.3.1 CVS with Remote Repositories

Before you can connect to a remote repository, you must:

- Have version control setup on your local machine
- Have version control setup on a remote server

- Have a working directory on your local machine that points to the repository on the server.

To set up CVS version control with a remote repository, follow the steps below:

1. Ask your system administrator to install CVS and create a remote repository with a module that you can use to control your Enterprise Architect package files. Your administrator must create a username and password for you before you can make a connection.
2. Open a command prompt window and navigate to, or create, a suitable directory to hold your CVS working copy directory; for example:
C:\> cd myCVSWorkSpace
3. Connect to the remote CVS repository. An example connection command is:
C:\myCVSWorkSpace> cvs -d:pserver:myUserID@ServerName:/repositoryFolder login

Note:

Replace myUserID with your CVS username, replace ServerName with the name of your CVS server and replace repositoryFolder with the path to the repository on the server.

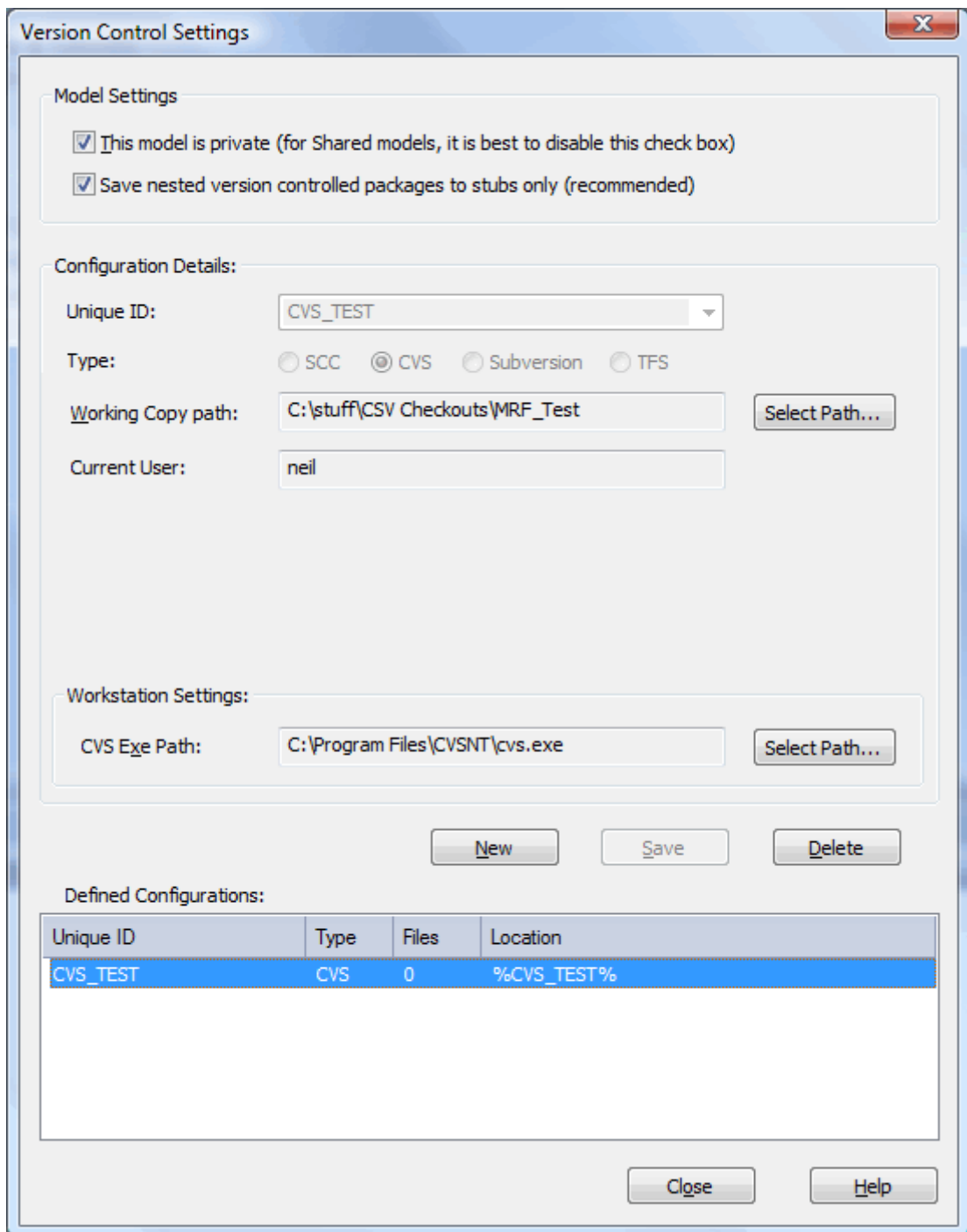
4. Create a local CVS workspace, derived from the remote repository. An example command is:
C:\myCVSWorkSpace> cvs -d:pserver:myUserID@ServerName:/cvs checkout moduleName

Note:

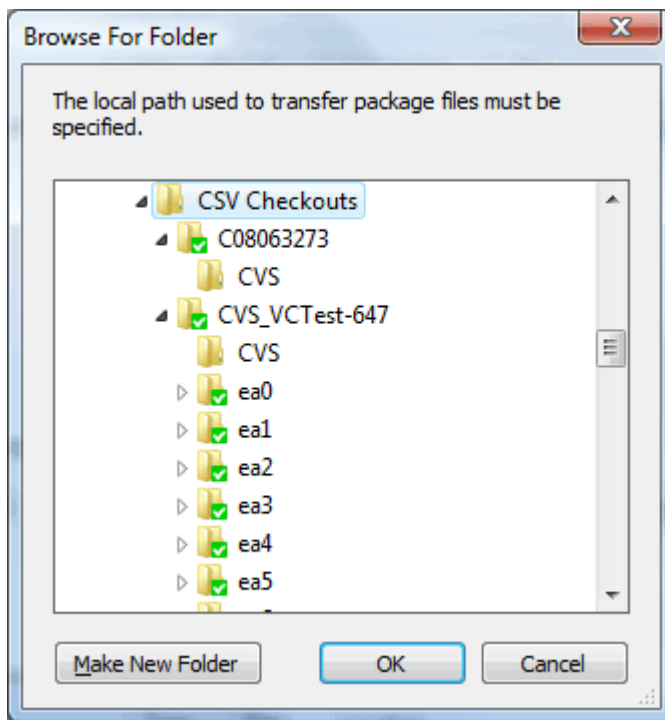
The above command creates a subdirectory in your current working directory, called moduleName. (Replace moduleName with the name of the module created by your system administrator). It creates local copies of all files contained in the CVS module found at ServerName:/cvs.

It also creates a subdirectory beneath moduleName, called CVS. This subdirectory contains a file called Root, that contains your CVS connection information. Enterprise Architect uses this file to obtain your CVS user ID.

5. Verify that your CVS installation is working correctly.
6. Change directory to the one you specified as the working copy, in the cvs checkout command above; that is, C:\myCVSWorkSpace\moduleName
7. Now create a test file, such as **Test.txt**, containing the text *CVS Test*. You can do this with the command:
echo CVS Test > Test.txt
8. Execute the following CVS commands:
 - cvs add Test.txt
 - cvs commit -m"Commit comment" Test.txt
 - cvs update Test.txt
 - cvs edit Test.txt
 - cvs editors Test.txt
9. The editors command should produce output resembling the following:
Test1.txt myUserID Tue Aug 9 10:08:43 2009 GMT myComputer C:\myCVSWorkSpace\moduleName
10. Take note of the userID that follows the filename. Enterprise Architect must find and use this user ID when you create your version control configuration. (See the example dialog below.)
11. Launch Enterprise Architect and open or create the model containing the packages to place under version control.
12. Select the **Project | Version Control | Version Control Settings** menu option. The **Version Control Settings** dialog displays.
13. Click on the **New** button, enter a suitable name in the **Unique ID** field, then click on the **CVS** radio button in the **Type** field.



- To specify the [Working Copy path](#) value, click on the **Select Path** button. Select the local folder in which to keep local working copies of the XML files to be stored in the Version Control repository.



15. Click on the **OK** button to return to the **Version Control Settings** dialog.
16. The **Current User** ^[19] field should display the user name used to log into the remote CVS repository. If this does not happen, it indicates that Enterprise Architect cannot extract the user name from the file `..\WorkingCopyPath\CVS\Root` and the configuration does not work correctly.
17. If necessary, set the **CVS Exe Path** ^[19] by clicking on the **Select Path...** button and browsing to the file path for the file `cvs.exe`, the CVS executable.
18. Click on the **Save** button to save the configuration you have defined. The new configuration is added to the list of **Defined Configurations**.

Note:

A new entry is also created in the *Local Paths* list, with the same ID as the new version control configuration. The **Local Path** entry records the Local Project path, for use in subsequent path substitutions. See the *Local Paths Dialog* topic in *Code Engineering Using UML Models*.

Options	Use to
This model is Private	Specify whether all users connect to a single shared copy of the model (such as a DBMS) or each user connects to their own private copy of the model. When unselected (for shared models), the option disables the File History - Retrieve functionality when the selected package is checked out by another user. This prevents modifications that might have been made by the other user from being discarded through importing a prior revision from version control.
Save nested version controlled packages to stubs only	Set nested version controlled packages to stubs or fully expanded trees. Defaults to selected. For a full explanation of this option, see the Version Control Nested Packages ^[11] topic.
Unique ID	Specify a configuration name that readily distinguishes it from other configurations. The unique ID displays as a selection in a list of Version Control configurations a package can connect to. In addition it is possible to select a previous version control configuration from this drop-down menu providing the configuration is not in use in the current model.

Options	Use to
Working Copy path	Specify the folder where the XML files representing the packages are stored. This folder should already exist before it is specified here. Every version control configuration you define in Enterprise Architect, should have its own local working copy folder in which to store working copies of the XMI package files; this should not be a shared network folder. Particularly bear this in mind if you are creating an Enterprise Architect project that is to be shared (e.g. a SQL database).
Current User	Specify the CVS user name associated with all CVS commands that are issued. This name is used by Enterprise Architect, to determine who has a package 'checked-out'.
CVS EXE Path	Specify the full path of the CVS client's executable file.

Note:

Sparx Systems strongly urge you not to manipulate version controlled package files outside of Enterprise Architect. It is possible to leave the package files in a state that Enterprise Architect cannot recognize.

5.3.2 CVS with Local Repositories

Before you can set up Enterprise Architect, you must have a working directory that points to a local repository; that is, one that is installed on your local machine. See your version control software help files for more information.

To set up CVS version control follow the steps below:

1. Launch Enterprise Architect and open or create the Enterprise Architect model for which packages are to be placed under version control.
2. Select the **Project | Version Control | Version Control Settings** menu option. The **Version Control Settings** dialog displays.
3. Click on the **New** button.
4. In the **Unique ID** field, type a suitable name for the configuration.
5. Against the **Type** field, click on the **CVS** radio button.

Model Settings

- This model is private (for Shared models, it is best to disable this check box)
- Save nested version controlled packages to stubs only (recommended)

Configuration Details:

Unique ID:

Type: SCC CVS Subversion TFS

Working Copy path:

Current User:

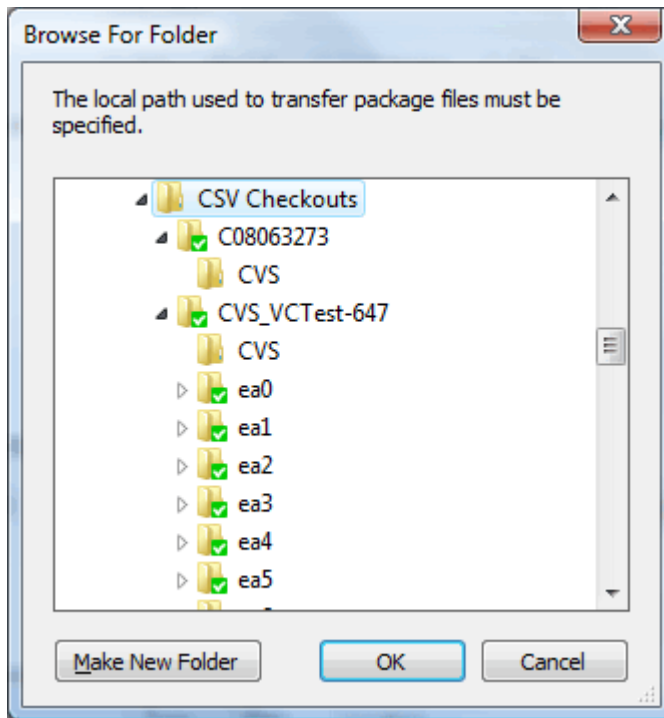
Workstation Settings:

CVS Exe Path:

Defined Configurations:

Unique ID	Type	Files	Location
CVS_TEST	CVS	0	%CVS_TEST%

- Click on the **Select Path...** button to the right of the **Working Copy path** field and browse for and select the local folder in which to keep local working copies of the XML files to be stored in the Version Control repository.
- If necessary, click on the **Select Path...** button to the right of the **CVS Exe Path** field and browse to the file path for the file `cvs.exe`, the CVS executable.



8. Click on the **Save** button to save the configuration you have defined.
9. The new configuration is added to the list in the **Defined Configurations** panel.

Note:

A new entry is also created in the Local Paths list, with the same ID as the new version control configuration. The **Local Path** entry records the Local Project path, for use in subsequent path substitutions. See the *Local Paths Dialog* topic in *Code Engineering Using UML Models*.

For further information on the fields in the **Version Control Settings** dialog, see the following table.

Field	Use to
This model is Private	Specify whether all users connect to a single shared copy of the model (e.g. a DBMS) or each user connects to their own private copy of the model. When unselected (for shared models), the option disables the File History - Retrieve functionality when the selected package is checked out by another user. This prevents modifications that might have been made by the other user from being discarded through importing a prior revision from version control.
Save nested version controlled packages to stubs only	Set nested version controlled packages to stubs or fully expanded trees. Defaults to selected. For a full explanation of this option, see Version Control Nested Packages .
Unique ID	Specify a name that readily distinguishes the configuration from other configurations. The Unique ID is displayed as a selection in the list of Version Control configurations a package can connect to. In addition it is possible to select a previous version control configuration from the drop-down menu providing the configuration is not in use in the current model.
Working Copy path	The folder where the XML files representing the packages are stored. This folder should already exist before it is specified here. Every version control configuration you define in Enterprise Architect, should have its own local Working Copy Folder in which to store working copies of the XML package files - this should not be a shared network folder. Particularly bear this in mind if you are creating an Enterprise Architect project that is to be

Field	Use to
	shared (for example, a SQL database).
Current User	The CVS user name associated with all CVS commands that are issued. This name is used by Enterprise Architect, to determine who has a package 'checked-out'.
CVS EXE Path	The full path name of the CVS client's executable file.

Note:

Sparx Systems strongly urge you not to manipulate version controlled package files outside of Enterprise Architect. It is possible to leave the package files in a state that Enterprise Architect cannot recognize.

5.4 Version Control with Subversion

Subversion is used to manage files and directories and is an open source version control system. To make use of Subversion control you must have Enterprise Architect version 6.0 or greater.

Tasks in setting up version control with Subversion include:

- [Set up Subversion](#) ^[22]
- [Create a new Repository Sub-Tree](#) ^[23]
- [Create a Local Working Copy](#) ^[24]
- [Subversion Under WINE-Crossover](#) ^[24]
- [Version Control Configuration](#) ^[26]
- [TortoiseSVN](#) ^[28]

Note:

To see a video demonstration of setting up a Subversion repository for version control, go to http://www.sparxsystems.com.au/resources/demos/settingupsubversion/svn_final.htm.

5.4.1 Set up Subversion

Obtain and Install Subversion

Note:

Enterprise Architect relies on exclusive file locking when applying version control to its packages. File locking was not introduced into Subversion until version 1.2. Enterprise Architect does not work with Subversion releases earlier than Subversion 1.2.

Before Enterprise Architect can be used with Subversion, the appropriate software must be installed by a Subversion administrator. Ask your system administrator to obtain and install the Subversion server and client applications.

Enterprise Architect must use the Subversion command line client to communicate with the Subversion server; it cannot use other clients such as [TortoiseSVN](#) ^[28].

Important:

Before you attempt to use Subversion through Enterprise Architect, you must first verify that you can use the Subversion command line client to access and operate on files within the working copy folder that Enterprise Architect will use. Your environment must be set up such that you can perform these operations without ever being prompted for user ID or password. For further information, please see the topic *Caching Client Credentials* in the official Subversion documentation.

The official Subversion documentation can be found at: <http://www.svnbook.red-bean.com/en/1.4/index.html>, while executable files for Subversion can be obtained from: http://www.subversion.tigris.org/project_packages.html#binary-packages.

You require the Windows executables for your client machines running Enterprise Architect in the windows environment. If you plan to run your Subversion server on a non-windows platform, you must download a binary suitable for that platform as well.

Chapter 6 in the Subversion documentation provides guidance on how to configure the server for different methods of access by the client. Secure connection methods are also covered in this chapter.

Your administrator should set up user IDs and passwords for every person who is to access the repository. Your administrator should then provide all users with the path to the repository, and ensure that they can all connect.

Before users can make use of Subversion, they must [create local working copies](#) ^[24] from the repository by checking-out a [repository sub-tree](#) ^[23].

Steps for setting up a repository and creating a local working copy can be found at: <http://www.svnbook.red-bean.com/en/1.4/svn.basic.in-action.html#svn.advanced.reposurls>.

Note:

Sparx Systems recommend that each new Enterprise Architect model being added to version control with Subversion should have a separate repository sub-tree created for it, and users should create a new local working copy from the sub-tree to be used with that model.

Repository URLs

Subversion repositories can be accessed using many different methods, on local disk or through various network protocols. A repository location, however, is always a URL. The table below describes how different URL schemas map to the available access methods.

Schema	Access Method
file:///	Direct repository access (on local disk).
http://	Access via WebDAV protocol to a Subversion-aware Apache server.
https://	Same as http://, but with SSL encryption.
svn://	Access via custom protocol to an svnserve server.
svn+ssh://	Same as svn://, but through an SSH tunnel.

For more information on how Subversion parses URLs, see <http://www.svnbook.red-bean.com/en/1.4/svn.basic.in-action.html#svn.advanced.reposurls>.

See Also

- [Configure Version Control with Subversion](#) ^[26]

5.4.2 Create a new Repository Sub-tree

If a repository sub-tree has already been created for your Enterprise Architect model, skip this topic and see the [Create a Local Working Copy](#) ^[24] topic. If your Enterprise Architect model has not previously been added to version control, create a sub-tree for it in your SVN repository by following the steps below:

1. Create a temporary directory structure to import into the SVN repository, which initializes the repository

sub-tree for this Enterprise Architect model. The directory structure should look like this:

```
tempDir
|
+--<EA_Model_Name>
|
|--trunk
|
+--branches
|
+--tags
```

2. Open a command prompt, navigate to *tempDir* and issue the command:

```
svn import. <repositoryURL> --message "A Comment of your choice"
```

Note:

After the import is finished, the original tree is not converted into a working copy. To start working, you must still **svn checkout** a fresh working copy of the tree.

3. Delete the directory *tempDir* and all its contents.

For further information see <http://www.svnbook.red-bean.com/en/1.4/svn.reposadmin.basics.html>.

5.4.3 Create a Local Working Copy

Once you have created a sub-tree in the repository for this model, or if one already exists, you are ready to create the local Working Copy for use with this model. Follow the steps below:

1. Choose a suitable directory on your system, in which to create your Subversion Working Copy. The directory that contains your model's .EAP file is probably a good choice.
2. Open a command line window, navigate to the directory to hold your Working Copy directory and check-out the model's sub-tree from the repository, with the following command:

```
svn checkout <repositoryURL>/<EA_Model_Name>,
```

where *<EA_Model_Name>* is the directory name that you used in setting up the repository sub-tree above.

After you have created your working copy, you should verify everything is working correctly before you attempt to use it from within Enterprise Architect. You must be able to commit files to the repository, without being prompted for ID or passwords.

Enterprise Architect interacts with Subversion using its command line client. Firstly, create a file in your working copy folder then, from a command prompt, add and commit the file to the repository. Use the following commands:

```
svn add <fileName>
svn commit <fileName> -m"A meaningful comment."
```

Now, update the file from the repository, lock the file, edit it and commit once more. Use the following commands:

```
svn update <fileName>
svn lock <fileName>
```

Then edit and save the file using your preferred editor:

```
svn commit <fileName> -m"A meaningful comment."
```

5.4.4 Subversion Under WINE-Crossover

When running Enterprise Architect under WINE/CrossOver, you can use either a Windows-based Subversion client or a Linux-based Subversion client.

If you intend to use the HTTP or HTTPS protocols you must use the Unix-like client, as the Windows client

cannot access the libraries necessary to create the required network connections. It is also easier to set up your working copy folder using the native Unix-like Subversion client and then continue using that same client from within Enterprise Architect.

However, to make use of the Unix-like client under CrossOver, you must also download and install a bridging utility called `SVN_gate`, which is available from the CodeWeavers' (CrossOver) web site:

<http://www.codeweavers.com/support/wiki/EASvn>

When using the Windows Subversion client, you simply install the Win32 Subversion client under CrossOver. Once you have set up your working copy directory, you are ready to use Subversion with Enterprise Architect. However, setting up your working copy is more difficult when using Win32 Subversion under CrossOver. Because you cannot see any output from Subversion commands run under CrossOver, the best way to run the commands is to create a Windows batch file containing the command to run, and to run that batch file as a Windows command under CrossOver. (See the [example batch file](#) ^[25] below.)

If you are running directly under WINE, launch the Windows batch file from a Unix shell script such as follows:

```
/home/user/cxoffice/bin/wine --bottle "ea" --untrusted
--workdir "/home/user/.cxoffice/ea"/drive_c"
-- "/home/user/.cxoffice/ea/drive_c/batfile.bat"
```

Enterprise Architect uses the Subversion command line client to communicate with your Subversion server. In order for Enterprise Architect to work successfully with Subversion, your Subversion working copy environment must be set up such that you can issue commands to Subversion from the command line, without ever being prompted for user input such as username or password.

By default, whenever the Subversion command-line client successfully responds to a server's authentication challenge, it saves the credentials in the user's private runtime configuration area, which is:

- `~/.subversion/auth/` on Unix-like systems or
- `%APPDATA%/Subversion/auth/` on Windows (which translates to `.../drive_c/windows/profiles/crossover/Application Data/Subversion/auth/` under CrossOver).

For this reason, Sparx Systems recommend that when you checkout a working copy from the Subversion repository, you specify your username and password on the command line, such that your credentials are cached from the outset. Use a Subversion command such as:

- Using a Unix-like client (run the command from the command line):

```
svn checkout --username "UserName" --password "myPassword" "svn://myServerName:3690/myProject"
"/.../drive_c/workingCopyDirectory"
```

- Using a Win32 client (run the command within a batch file run under CrossOver):

```
"C:\Program Files\Subversion\bin\svn.exe" checkout --username "UserName" --password "myPassword"
"svn://myServerName:3690/myProject" "C:\SVN-test\workcopy" >"C:\SVN-test\stdout.txt" &2>"C:\SVN-test\stderr.txt"
```

It is a good idea to checkout your Subversion working copy into a folder within the WINE bottle where Enterprise Architect is to run. In this way, the pathnames used for your version controlled package files are much shorter.

If you intend to use the Win32 Subversion client with Enterprise Architect, you should create the local working copy that Enterprise Architect is to use, by performing a Subversion checkout command using the Win32 client under WINE/CrossOver. Similarly, if you plan to use the Unix-like Subversion client with Enterprise Architect, you should perform the initial checkout using that client. In this way, your user credentials are cached in the correct location for the client that Enterprise Architect is using.

It is important to verify that your command line client for Subversion is working correctly before attempting to connect from Enterprise Architect. For guidance on verifying your set up, see the [Create a Local Working Copy](#) ^[24] topic.

The following is an example of a Windows batch file that can be used under CrossOver to run Subversion commands. Simply uncomment the command to execute. Each command should be a single line - the `\` is intended as a continuation character.

```
rem "C:\Program Files\Subversion\bin\svn.exe" checkout --username "UserName" --password "myPassword" \
"svn://myServerName:3690/myProject" "C:\SVN-test\workcopy"
>"C:\SVN-test\stdout.txt" &2>"C:\SVN-test\stderr.txt"
```

```
rem "C:\Program Files\Subversion\bin\svn.exe" add "C:\SVN-test\workcopy\myTestFile.xml" \
>"C:\SVN-test\stdout.txt" &2>"C:\SVN-test\stderr.txt"

rem "C:\Program Files\Subversion\bin\svn.exe" commit -m"a message" "C:\SVN-test\workcopy\myTestFile.xml" \
>"C:\SVN-test\stdout.txt" &2>"C:\SVN-test\stderr.txt"

rem "C:\Program Files\Subversion\bin\svn.exe" lock "C:\SVN-test\workcopy\myTestFile.xml" \
>"C:\SVN-test\stdout.txt" &2>"C:\SVN-test\stderr.txt"
```

5.4.5 Version Control Configuration

This topic assumes that you have already installed Subversion (both the server and the client parts), and that you have a [local working copy](#)^[24], derived from a [repository sub-tree](#)^[23], already set up for use with your Enterprise Architect model. If this is not the case, please see the [Set up Subversion](#)^[22] topic.

Once you have set up and tested the Local Working Copy, you are ready to define a Version Control configuration for use with the Enterprise Architect model to place under version control.

To apply version control to your Enterprise Architect model using the Subversion working copy that you have set up, follow the steps below:

1. Launch Enterprise Architect and open the model for which this Working Copy was created.
2. Select the **Project | Version Control | Version Control Settings** menu option.
3. Click on the **New** button, enter a suitable name in the **Unique ID** field, then click on the **Type: Subversion** radio button.

Model Settings

This model is private (for Shared models, it is best to disable this check box)

Save nested version controlled packages to stubs only (recommended)

Configuration Details:

Unique ID:

Type: SCC CVS Subversion TFS

Working Copy path:

Workstation Settings:

Subversion Exe Path:

Defined Configurations:

Unique ID	Type	Files	Location

4. To the right of the **Working Copy path** field, click on the **Select Path** button and select the local folder in which to keep local working copies of the XML files to be stored in the Version Control repository.
5. In the **Workstation Settings** panel, click on the **Select Path** button to specify the path for your Subversion client executable.
6. Click on the **Save** button to save the configuration you have defined; the new configuration is added to the **Defined Configurations** list.

Note:

A new entry is also created in the Local Paths list, with the same ID as the new version control configuration. The **Local Path** entry records the Local Project path, for use in subsequent path substitutions. See the *Local Paths Dialog* topic in *Code Engineering Using UML Models*.

7. When you have finished defining your version control configurations, click on the **Close** button.

Additional Information on the dialog fields:

Option	Use to
This model is private	Specify whether all users connect to a single shared copy of the model (such as a DBMS) or each user connects to their own private copy of the model. When unselected (for shared models), the option disables the File History - Retrieve functionality when the selected package is checked out by another user. This prevents modifications that might have been made by the other user from being discarded through importing a prior revision from version control.
Save nested version controlled packages to stubs only	Set nested version controlled packages to stubs or fully expanded trees. Defaults to selected. For a full explanation of this option, see the Using Nested Version Control Packages topic.
Unique ID	Specify a configuration name that readily distinguish this configuration from other configurations. The Unique ID displays as a selection in the list of Version Control configurations a package can connect to. In addition you can select a previous version control configuration from this drop-down menu, providing the configuration is not in the current model.
Working Copy path	Specify the folder where the XML files representing the packages are stored. This folder should already exist before it is specified here. Every PC using Subversion version control should have its own Subversion working copy folder in which to store working copies of the XMI package files; this should not be a shared network folder. Particularly bear this in mind if you are creating a .EAP file that is to be shared (for example, a SQL database).
Subversion Exe Path	Specify the full path name of the Subversion client executable file.

Note:

Sparx Systems strongly urge you not to manipulate version controlled package files outside of Enterprise Architect. It is possible to leave the package files in a state that Enterprise Architect cannot recognize.

5.4.6 TortoiseSVN

TortoiseSVN is a Windows shell extension for Subversion.

Enterprise Architect cannot use TortoiseSVN to communicate with the Subversion server; it must use the Subversion command line client.

TortoiseSVN provides icon overlays in Windows Explorer that are useful as a tool for observing the status of your Subversion controlled files. It enables you to create your repository sub-trees and check out local working copies from within Windows Explorer using simple menu commands.

Note:

- Sparx Systems recommend that you test your local working copies, by adding and committing a dummy file from the command prompt window.
- Manipulating Enterprise Architect's package files, using tools that are external to Enterprise Architect, could leave those files in a state that Enterprise Architect cannot use.

You can download TortoiseSVN from: <http://www.tortoisesvn.tigris.org/>.

5.5 Version Control with TFS

In order to use Team Foundation Server (TFS) for version control with Enterprise Architect, all users must have either the TFS command line client (*tf.exe*) or Microsoft's Team Foundation Server MSSCCI installed on their local machine. Each intended user must also have an account that provides read/write access to a

workspace on the server.

This topic covers configuring version control using the TFS command line client. To configure version control with the TFS MSSCCI client, please follow the instructions in the [Version Control with SCC](#)^[11] topic.

The following preliminary steps should be performed within TFS on each PC and for each user, before making any attempt to [define a Version Control Configuration within Enterprise Architect](#)^[29] that uses TFS.

Each user must set up a separate workspace for use in version control in Enterprise Architect, containing a single local working folder on their own machine that is mapped to a Source Control folder on the server.

When initializing the connection to TFS, Enterprise Architect issues the command *tf get*. If the *Local Working Copy* path specified in Enterprise Architect's version control configuration is mapped through a TFS workspace that also maps many *other* working folders to their corresponding Source Control folders, TFS can take a long time as it proceeds to update the files in all of those folders. Enterprise Architect might appear to freeze when it initializes the connection to TFS, whilst it waits for TFS to complete the *tf_get* command and hand back program control.

Note:

To see a video demonstration of setting up a TFS project for version control, go to <http://www.sparxsystems.com.au/resources/demos/settinguptfs/TFS%20Project-Workspace%20Setup.htm>.

5.5.1 Connect an Enterprise Architect Model to Version Control using TFS

Having set up TFS and created or otherwise opened your model, you can configure the model for version control under TFS. To do this, follow the steps below:

1. Open or create the Enterprise Architect model to place under version control.
2. Select the **Project | Version Control | Version Control Settings** menu option. The **Version Control Settings** dialog displays.
3. Click on the **New** button, in the **Unique ID** field enter a suitable name, then select the **TFS** radio button.

Model Settings

This model is private (for Shared models, it is best to disable this check box)

Save nested version controlled packages to stubs only (recommended)

Configuration Details:

Unique ID: TFS-config

Type: SCC CVS Subversion TFS

Working Copy path: C:\VC Workspaces\TFS\WorkFolder1

Server Name:

Workspace Name:

User Name: SparxSystems\userOne

Password: ●●●●●●●●

Workstation Settings:

TFS Exe Path: files\Microsoft Visual Studio 8\Common7\IDE\tf.exe

Defined Configurations:

Unique ID	Type	Files	Location
-----------	------	-------	----------

- Click on the **Select Path...** button to the right of the **Working Copy path** field, and select the local folder in which to keep local working copies of the XML files to be stored in the Version Control repository.

Note:

Enterprise Architect queries TFS to retrieve the Server and Workspace names associated with this folder, when attempting to save the configuration data.

- In the **User Name** and **Password** fields, type values that enable access to the TFS workspace associated with the Working Copy path specified above.

Note:

Users who automatically log in to TFS through means external to Enterprise Architect (for example, through MS Integrated Security) can leave the **User Name** and **Password** fields blank. If the **Password** field is blank, Enterprise Architect retrieves the current user's Windows username and uses that value when determining whether a package is checked out to them or to some other user.

6. The **TFS Exe Path** field displays the default installation path. Click on the **Select Path...** button if it is necessary to modify this field.
7. Click on the **Save** button to save the configuration you have defined.
8. The new configuration is added to the list in the **Defined Configurations** panel.

Note:

A new entry is also created in the Local Paths list, with the same ID as the new version control configuration. The **Local Path** entry records the Local Project path, for use in subsequent path substitutions. See the *Local Paths Dialog* topic in *Code Engineering Using UML Models*.

9. When you have finished defining your version control configurations, click on the **Close** button.

Additional Information on the Dialog Fields

Option	Use to
This model is private	Specify whether all users connect to a single shared copy of the model (such as a DBMS) or each user connects to their own private copy of the model. When unselected (for shared models), the option disables the File History - Retrieve functionality when the selected package is checked out by another user. This prevents modifications that might have been made by the other user from being discarded through importing a prior revision from version control.
Save nested version controlled packages to stubs only	Set nested version controlled packages to stubs or fully expanded trees. Defaults to selected. For a full explanation of this option, see Use Nested Version Control Packages ^[11] .
Unique ID	Specify a configuration name that readily distinguishes it from other configurations. The Unique ID is added to the list of Version Control configurations a package can connect to. In addition it is possible to select a previous version control configuration from this drop-down menu providing the configuration is not in the current model.
Working Copy Path	Specify the folder where the XML files representing the packages are stored. This folder should already exist before it is specified. Every PC using TFS version control should have its own TFS Local Folder in which to store working copies of the XMI package files - this should not be a shared network folder. Particularly bear this in mind if you are creating a .EAP file which is to be shared (for example, a SQL database).
Server Name	Specify the name of the Team Foundation Server to connect to.
Workspace Name	Specify the name of a pre-defined TFS workspace that you are using.
User Name	Specify the user name that you use to connect to the Team Foundation Server. The user name that you specify should give you read/write permissions in the specified workspace.
Password	Specify the password associated with the user name you specify. Enterprise Architect stores this password, in encrypted form, as part of the version control configuration data.

Option	Use to
TFS Exe Path	Browse to and select the full path name of the TFS command line client's executable file.

Notes:

- Sparx Systems strongly urge you not to manipulate version controlled package files outside of Enterprise Architect. It is possible to leave the package files in a state that Enterprise Architect cannot recognize.
- Visual Studio Integration (MDG Integration for Visual Studio 2005 or 2008) enhances TFS support by providing access to, for example, work items and bugs within both Enterprise Architect and the MDG Integration product.

6 Use Version Control

The following topics describe the most common activities using the version control features of Enterprise Architect, accessed through the [Package Version Control Menu](#)^[33]:

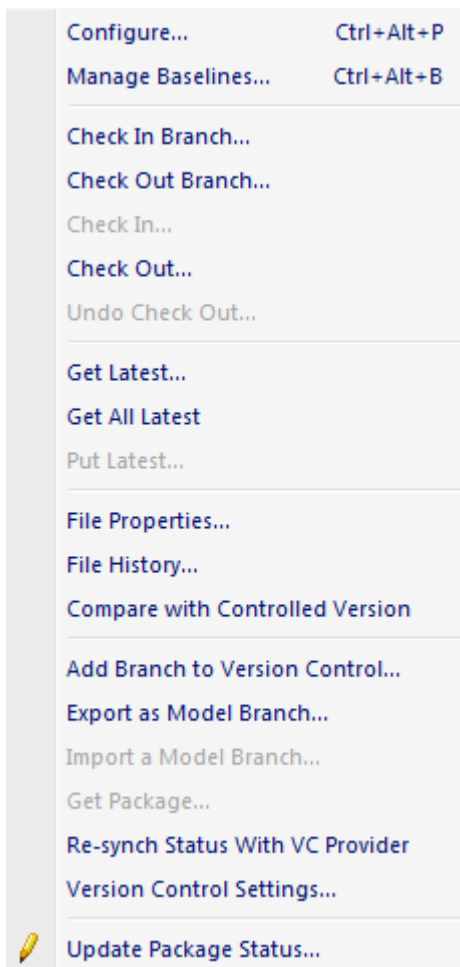
- [Configure Controlled Package](#)^[36]
- [Use Existing Configuration](#)^[37]
- [Validate Package Configurations](#)^[37] (**Project | Version Control** menu)
- [Check In and Check out Packages](#)^[38]
- [Include Other Users Packages](#)^[40]
- [Apply Version Control to Branches](#)^[41]
- [Export Controlled Model Branch](#)^[42]
- [Import Controlled Model Branch](#)^[42]
- [Review Package History](#)^[44]
- [Refresh View of Shared Project](#)^[44]

General Notes

- The export/import facility is not fast and submitting packages containing many sub-nodes to version control should be avoided. It is recommended version control is applied to individual packages. See the [Version Control Nested Packages](#)^[11] topic for more information.
- Replication should not be combined with version controlled packages. (See the *Replication* topic in *UML Model Management*.)
- Sparx Systems strongly urge you not to manipulate version controlled package files outside of Enterprise Architect. It is possible to leave the package files in a state that Enterprise Architect cannot recognize.

6.1 Package Version Control Menu

To display the **Version Control** menu, right-click on a version controlled package in the **Project Browser** and select the **Package Control** context menu option. (This menu displays a number of different options if the selected package is not under version control - see the *Controlled Package Menu* topic in *UML Model Management*.)



Menu Option & Function Keys	Use to
Configure [Ctrl]+[Alt]+[P]	Display the Package Control Options ^[36] dialog which enables you to specify whether this package (and its children) is controlled, which file it is controlled through, and which version control configuration to use.
Manage Baselines [Ctrl]+[Alt]+[B]	Create a Baseline of the current package, or compare the current package with a previous Baseline.
Check In Branch	For the selected branch of the model, (that is, the selected package and all of its child packages) display the Select Packages to Check In ^[39] dialog, listing all version controlled packages within that branch that are checked out to you. You can then select packages in the displayed list, to be submitted for check-in.
Check Out Branch	For the selected package, check out the package and recursively check out all of its contained sub-packages ^[40] . Retrieves the latest version of the packages from the central repository, overwriting the current packages. After check out, the packages are available for editing.
Check In	Submit the currently selected package to the central repository. Enterprise Architect prompts you to enter optional comments describing changes to the packages.
Check Out	Retrieve the latest version of the currently selected package from the central repository, overwriting the current packages. After check out the

Menu Option & Function Keys	Use to
	packages are available for editing.
Undo Check Out	Cancel all changes you have made to the currently selected package. This restores the model to the state it was in before the package was checked out, leaving the select package and sub-packages locked.
Get Latest	Retrieve the latest revision of the package from the repository. Available only for packages that are checked in.
Get All Latest	Retrieve the latest revision of all version controlled packages in the project. Only retrieves packages that are checked in.
Put Latest	Update the central repository with the currently selected package (which you have checked out), while retaining checkout status on the package. This is equivalent to checking a package in and immediately checking it back out again.
File Properties	Ask the version control provider to show the version control properties associated with the XML export file pertaining to the currently selected package. This also identifies who has checked out the package.
File History	Where the controlling package has been configured by an SCC provider, this provider shows a change history for the package. See your provider's documentation for details on how to use the control. Otherwise, if the version control is CVS, the history is shown via Enterprise Architect's internal CVS history menu.
Compare with version on disk	Compare the current package with the XMI version on disk.
Add Branch to Version Control	Apply version control to all packages within a selected <i>model branch</i> , in a single operation. In this context, a model branch is a package that is currently selected in the Project Browser , and all of the packages contained within it.
Export as Model Branch	Export ^[42] a newly created model branch from your own private copy of a model.
Import a Model Branch	Retrieve ^[42] a model branch and import it into either the source model or another model.
Get Package	Access packages in the version control repository that are not currently available in your model.
Re-synch Status With VC Provider	Update the version control status value recorded for the selected package in the Enterprise Architect project to match the value reported by the version control provider ^[45] , without performing an XMI import or export. Use this function when the package's version control status recorded in your Enterprise Architect project is out of synchrony with the version control status reported by your version control provider.
Version Control Settings	Display the Version Control Settings dialog ^[9] .
Update Package Status	Provide a bulk update on the status of a package, including status options such as Proposed , Validate and Mandatory . Note: This option is a generic package option not specific to version control.

6.2 Configure Controlled Package

Before working on a package under version control, you must define it as a controlled package and specify the version control configuration to use.

Note:

In the Corporate, Business and Software Engineering, System Engineering and Ultimate editions of Enterprise Architect, if security is enabled you must have **Configure Packages** permission to configure packages for version control. See *User Security in UML Models*.

Configure a Version Controlled Package

To configure a version controlled package, follow the steps below:

1. In the **Project Browser**, right-click on the package to place under version control. The context menu displays.
2. Click on the **Package Control | Configure** menu option. The **Package Control Options** dialog displays.

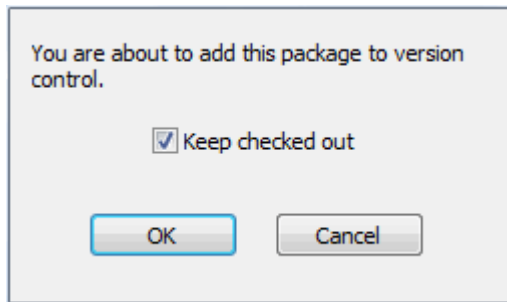
The screenshot shows the 'Package Control Options' dialog box. It contains the following fields and options:

- Control Package:**
- Version Control:** (None) (dropdown menu)
- XMI Filename:** Resources.xml (text field with browse button)
- UML/XMI Type:** Enterprise Architect XMI/UML 1.3 (dropdown menu)
- Version ID:** 1.0 (text field)
- Owner:** Nigel Pitt (dropdown menu)
- Last Load Date:** (empty text field)
- Last Save Date:** (empty text field)
- Other Options:**
 - Use DTD
 - Log Import/Export
 - Batch Import
 - Batch Export
 - Include sub-packag

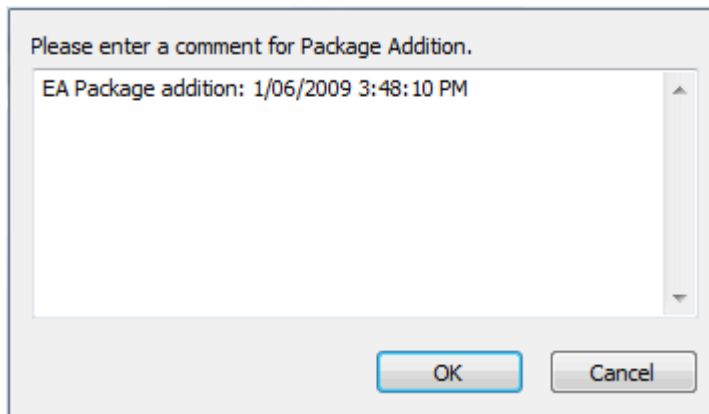
Buttons: OK, Cancel, Help

3. Select the **Control Package** checkbox to indicate that this is a controlled package.
4. Click on the **Version Control** drop-down arrow and select the version control repository; this connects the package to a specific version control configuration.

The **XMI Filename** field then displays the version control configuration default path.
5. The **Version ID** field defaults to **1.0**; if necessary, change this to the appropriate reference.
6. The **Owner** field defaults to your user name; if necessary, type or select the name of the user who owns the package.
7. Click on the **OK** button to set the version control options. The **Add Package to Version Control** dialog displays.



8. If you do not want to check-out the package immediately, clear the **Keep checked out** checkbox.
9. Click on the **OK** button. The **Add Comment** window displays.



This window displays the date and time at which the package was put under version control.

10. If required, type any further comments in the window. Click on the **OK** button.

Enterprise Architect places the package under the version control configuration you selected, and marks the package in the **Project Browser** with the version controlled [checked out or not checked out](#) icons, as appropriate.

6.3 Use Existing Configuration

Once a version control configuration has been defined in one model it is possible to add the configuration to other models. To use this feature follow the steps below:

1. Open the model that is to have the predefined version control configuration added to it.
2. Right-click on any package in the **Project Browser** and select the **Package Control | Version Control Settings** context menu option. The [Version Control Settings](#) dialog displays.
3. Click on the **New** button.
4. In the **Unique ID** field, click on the drop-down arrow and select one of the previously-defined version control configurations.
5. Click on the **Save** button to confirm the version control configuration.

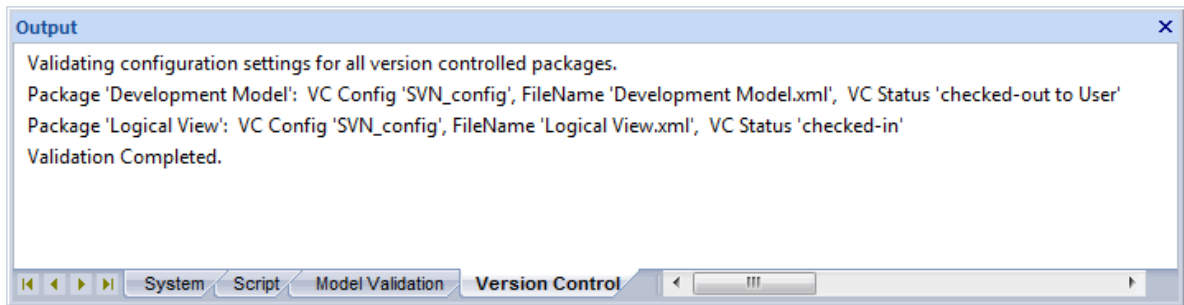
6.4 Validate Package Configurations

You can test the validity of the version control settings associated with each version controlled package within your current model. To do this, select:

Project | Version Control | Validate Package Configurations

The validation process scans the model database and verifies that the version control configuration associated with each version controlled package is fully specified in the current model. It also queries the corresponding version control provider to find the status of the package file associated with each version controlled package.

The results of the validation process are sent to the Enterprise Architect **Output** window, as shown below:



Depending on the results, you can then complete the definition of any invalid or missing version control configurations, or correct problems with individual packages or their associated package files.

Click on an error message to highlight, in the **Project Browser**, the package that is in error.

6.5 Check In and Check Out Packages

To work on a version controlled package you must have the package checked out. When a package is checked out to a specific user, a write lock is set on the package and other users cannot make changes to it until it has been checked in again.

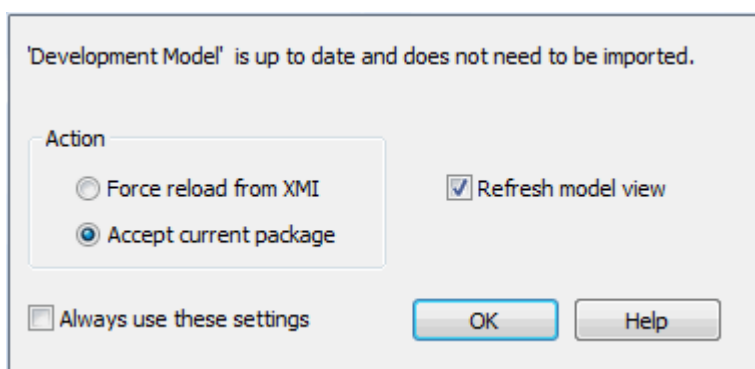
Note:

In the Corporate, Business and Software Engineering, System Engineering and Ultimate editions of Enterprise Architect, if security is enabled you must have **Use Version Control** permission to check files in and out using version control. See *User Security in UML Models*.

Check In/Check Out

1. In the **Project Browser**, right-click on the package icon.
2. Select the **Package Control | Check In, Check Out** or **Undo Checkout** context menu options, as appropriate.
3. If required, enter a comment when prompted to do so.

If you are working in a **private** model and you select the **Check Out** menu option, the **Import Package** dialog displays (in shared models, this dialog only has default values and therefore does not display).



Option	Use to
Force reload from XMI	Reload the package content from the XMI file in the central repository, even though the package and XMI file are synchronized. This ensures that links and dependencies that might not have been refreshed are updated as well.
Accept current package	(The default.) Leave the package content in its current state.

Option	Use to
Refresh model view	Refresh the model view to show any changes from other checked out packages.
Always apply above settings	Apply the settings in the above three fields every time you check out a package that is found to be up to date, and therefore do not display this dialog again. Note: To display the dialog if, for example, you want to change the settings, press [Ctrl] while you select the Package Control Check Out menu option.

The package icon in the **Project Browser** should change. When you check out a package this is represented by a figure 8 to the left of the package icon. When you check in a package the package icon is overlaid with a colored rectangle and key. In the example below, the upper package is checked out whilst the lower package is checked in.

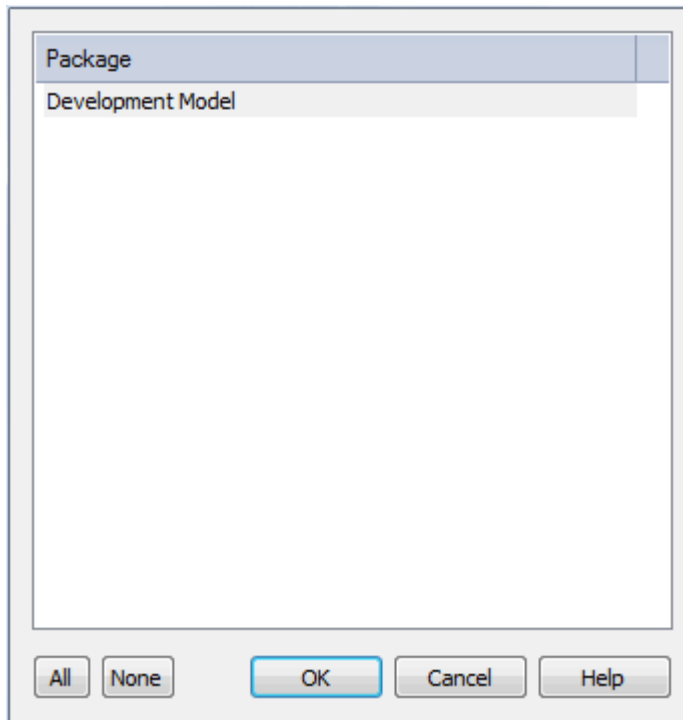


Notes:

- If you check out a version controlled package whilst offline, the package icon has a red figure 8 in front of it. See [Offline Version Control](#) [46].
- If the packages under version control contain any alternative images (See *UML Modeling With Enterprise Architect - UML Modeling Tool*) and those images are subject to **frequent change**, you can set the **Export alternate images** option on the **Options** dialog (See *Using Enterprise Architect - UML Modeling Tool*) to export the images to the version control repository when you check in the packages. If the images are not subject to frequent change, do not select this option and instead use Export/Import Reference Data to manage alternative images (See *UML Model Management*).

Check In Branch

1. In the **Project Browser**, right-click on the package icon at the root of the model branch that is to be checked in and select the **Package Control | Check In Branch** context menu option. The **Select Packages to Check-in** dialog displays, listing all version controlled packages within the branch that you have checked out.



2. Click on the package to check in, or use:
 - **[Ctrl]+click** to add or remove several individual packages
 - **[Shift]+click** to select a range of packages
 - **All** to select all packages listed
 - **None** to clear all selected packages.
3. Click on the **OK** button to check-in the selected packages.
4. If required, enter a comment when prompted to do so. (This comment applies to all packages that you have checked in.)
5. Each package icon changes to indicate that the packages have been checked-in.

Check Out Branch

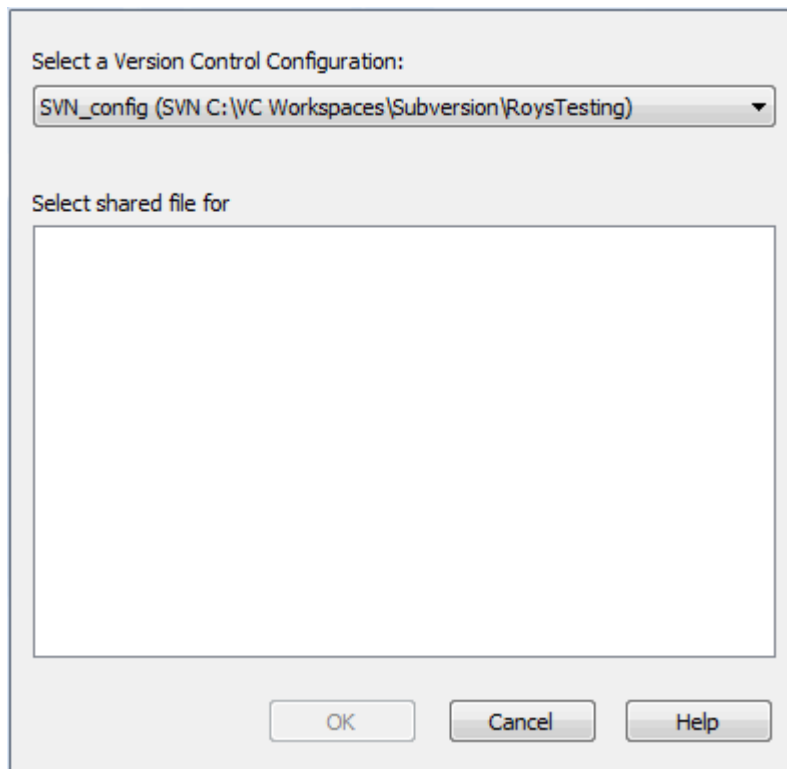
1. In the **Project Browser**, right-click on the name of the root package of the model branch to be checked out and select the **Package Control | Check Out Branch** context menu option. The selected package and all of its contained sub-packages are recursively checked out.
2. Any packages that cannot be checked-out are listed in a message box, with a brief description of the problem. For example: *The package is already checked out by user 'Fred'*.
3. When Project Security is enabled in *Lock to Edit* mode, Enterprise Architect prompts you to apply a User Lock throughout the selected model branch before proceeding.

6.6 Include Other Users' Packages

You can retrieve packages that have been created by other users, or by you in another model, from version control and import them into your current model.

Other users might be creating packages to use in your model. If you are not sharing a SQL database or .EAP file, those packages do not automatically become part of your model. If the packages have been placed into version control, you can retrieve them and import them into your model as children of an existing package, using the **Get Package** command.

1. You must have access to the package files through the version control system and you must define a Version Control Configuration through which to access those files. The version control configuration must use the same unique ID that was originally used to add the package to version control.
2. In the **Project Browser**, right-click on the package to use as the parent of the incoming package.
3. Select the **Package Control | Get Package** context menu option. The **Get Shared File** dialog displays.



4. In the **Select a Version Control Configuration** field, click on the drop-down arrow and select the version control configuration associated with the package to retrieve. The file list is populated with the names of files available through that configuration, for retrieval and import into your model.
5. Select the package file to import into your model and click on the **OK** button.

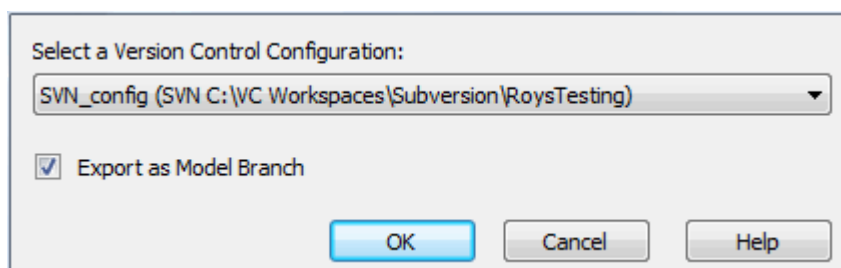
6.7 Apply Version Control To Branches

Enterprise Architect enables you to apply version control to all packages within a selected *model branch*, in a single operation. In this context, a model branch is a package that is currently selected in the **Project Browser**, and all of the packages contained within it.

The [Version Control Configuration](#) ^[9] to be used in this operation must be defined within the model before selecting this command.

To apply version control to a model branch, follow the steps below:

1. Right-click on the required package and select the **Add Branch to Version Control** ^[33] context menu option. The **Apply VC to Branch** dialog displays.



2. In the **Select a Version Control Configuration** field, click on the drop-down arrow and select the Version Control Configuration to use.
3. If required, select the **Export as Model Branch** checkbox to [export the selected package \(and sub-packages\) as a Model Branch](#) ^[42].

- Click on the **OK** button. Enterprise Architect creates a number of sub-folders within the version control working copy folder, before exporting all of the packages within the selected model branch. Enterprise Architect generates package filenames using the package GUIDs, before adding the resulting files to version control.

If you have selected the **Export as Model Branch** checkbox, once the version control operation is complete Enterprise Architect also creates a Model Branch file (.EAB file). You can subsequently [import the version-controlled Model Branch](#) ^[42].

6.8 Export Controlled Model Branch

You might want to export a newly created model branch from your own private copy of a model so that, for example:

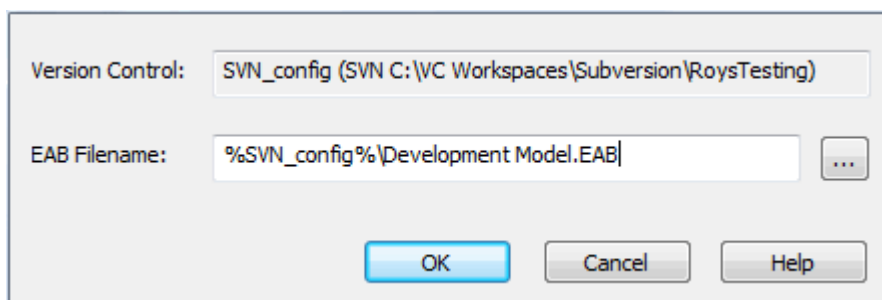
- Another user can import that branch into their own private copy of the same model, or
- It can be imported for inclusion as a common branch in a number of different models.

Applying version control to an Enterprise Architect model can result in many XML files placed under version control. It could then be hard to locate and import the file corresponding to the root of a particular model branch. Enterprise Architect's Model Branch Files (.EAB files) overcome this problem by simplifying the retrieval of model hierarchies for use in other models.

The facility is only enabled for packages that are already under version control. The exported Model Branch File is also placed under version control, using the same version control configuration that is controlling the selected package.

To export a model branch, follow the steps below:

- Right-click on the version controlled package to export as a model branch.
- Select the **Package Control | Export as Model Branch** context menu option. The **Export as Model Branch** dialog displays.



- In the **EAB Filename** field, type a name for your Model Branch File. Alternatively, click on [...] and browse for the file location.

Note that the package name is supplied as a default.

You can specify any file name, including sub-folder names, as long as the file is contained in or below the working folder of your version control configuration.

6.9 Import Controlled Model Branch

It might be necessary to either:

- Retrieve a model branch created by another user in a private copy of a model, to import it into your own private copy of the same model or
- Retrieve a model branch that is common in many models, for inclusion in a new model.

Applying version control to an Enterprise Architect model can result in many XML files placed under version control. It could then be hard to locate and import the file corresponding to the root of a particular model branch. Enterprise Architect's Model Branch files overcome this problem by simplifying the retrieval of model hierarchies for use in other models.

The **Import a Model Branch** context menu option uses Enterprise Architect's Model Branch Files, of which there are few, to retrieve information about the root package file and import the model branch. The Model

Branch File records information such as the name and type of the version control configuration for the selected package, and the relative filename of the version controlled XMI file associated with the package.

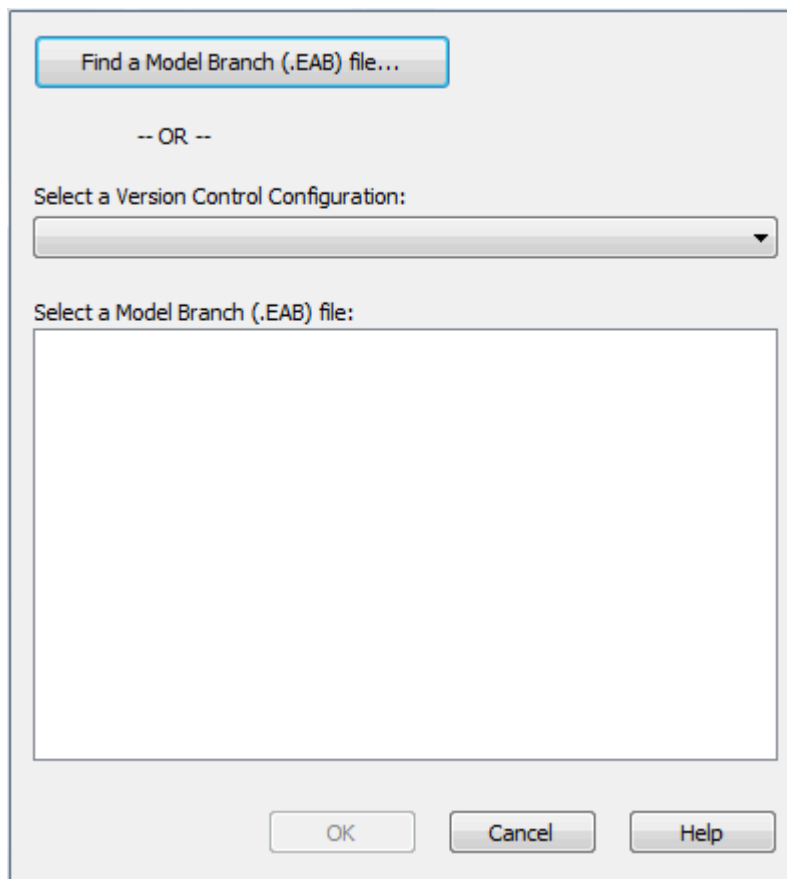
Before attempting to import a model branch, you must have access to the version controlled XMI files that represent the model branch to be imported. That is, there must be a working copy folder, accessible from the machine on which Enterprise Architect is running, that is associated with the Version Control repository containing those XMI files.

It is not necessary to have the relevant Version Control Configuration set up within Enterprise Architect before issuing this command - Enterprise Architect prompts you to complete specification of the configuration if necessary.

The **Import a Model Branch** context menu option is only enabled for packages that you (the current user) are able to edit, as the imported model branch is inserted into the model under the selected package.

To import a model branch, follow the steps below:

1. Right-click on the package into which the model branch is to be imported.
2. Select the **Package Control | Import a Model Branch** context menu option. The **Import VC Model Branch** dialog displays.



3. Either:
 - Click on the **Find a Model Branch (.EAB) file** button and browse for the Model Branch File. If the version control configuration used by the file has not been fully set up, Enterprise Architect prompts you to complete and save the configuration. The model branch import then proceeds. OR
 - If the version control configuration used by the file has been fully set up in the current model, click on the drop-down arrow in the **Select a Version Control Configuration** field and select the configuration, then select the Model Branch File from the **Select a Model Branch (.EAB) file** list. Click on the **OK** button to import the model branch.

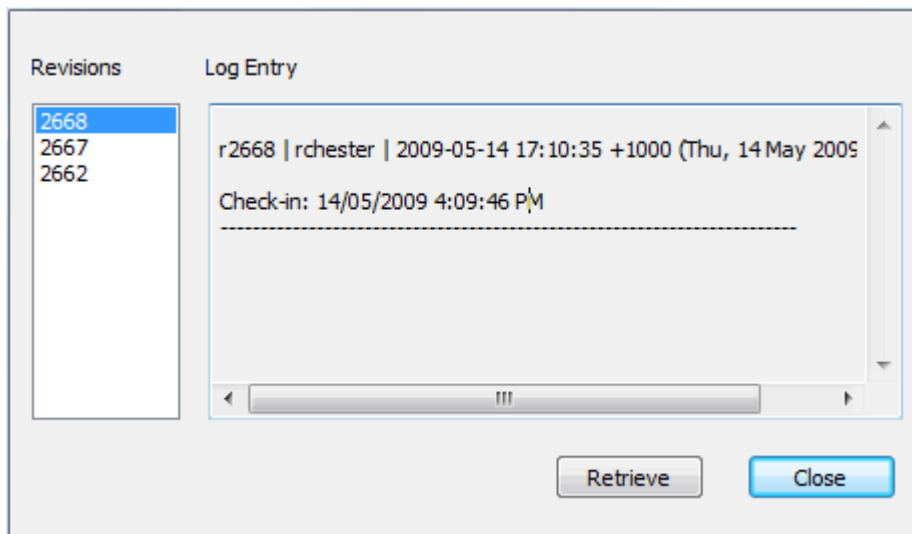
Enterprise Architect imports the root package specified in the Model Branch File and recursively imports and populates all the sub-packages contained in the root package.

6.10 Review Package History

Reviewing package history enables you to view the history of checked in package revisions. It also enables you to import selected prior revisions of the package into your model. The package revision is retrieved in read-only form, enabling you to view the package contents but not make any changes to the package.

To review package history follow the steps below:

1. In the **Project Browser**, right-click on the package configured for version control. The context menu displays.
2. Select the **Package Control | File History** menu option.
 - If the package has been configured through SCC, you access the history through the mechanism offered by the third party SCC provider.
 - If the package has been configured for CVS, Subversion or TFS, Enterprise Architect's **File Version History** dialog displays and the following steps apply.



3. In the **Revisions** field, click on a revision number to view the log entries for that revision.
4. To view the package history select a revision and then click on the **Retrieve** button. A warning dialog displays, indicating that the package is being opened in read-only mode.
5. Click on the **Yes** button to continue or the **No** button to cancel the action. The package is retrieved in read only mode, enabling you to view the history of the package at the specified version.
6. To go back to the latest version, in the **Project Browser** right-click on the package and select either the **Package Control | Check Out..** context menu option or the **Package Control | Get Latest** context menu option.

6.11 Refresh View of Shared Project

When a user of a shared model checks out a package and makes changes, other users can see those changes by refreshing their view of the package or the changed diagram within the package.

You can refresh your view of the **Project Browser** in the following ways:

- Right-click on the package name in the **Project Browser** and select the **Contents | Reload Current Package** context menu option
- Select the **File | Reload Current Project** menu option (or select the **Reload Project** icon in the **Project** toolbar, or press **[Ctrl]+[Shift]+[F11]**)
- Close the project and reopen it.

You can refresh the current diagram in the following ways:

- Select the **Window | Reload Current View** menu option
- Right-click on the opened diagram tab in the diagram view, and select the **Reload <diagram name>** context menu option.

(For further information on these methods, see *Using Enterprise Architect - UML Modeling Tool*.)

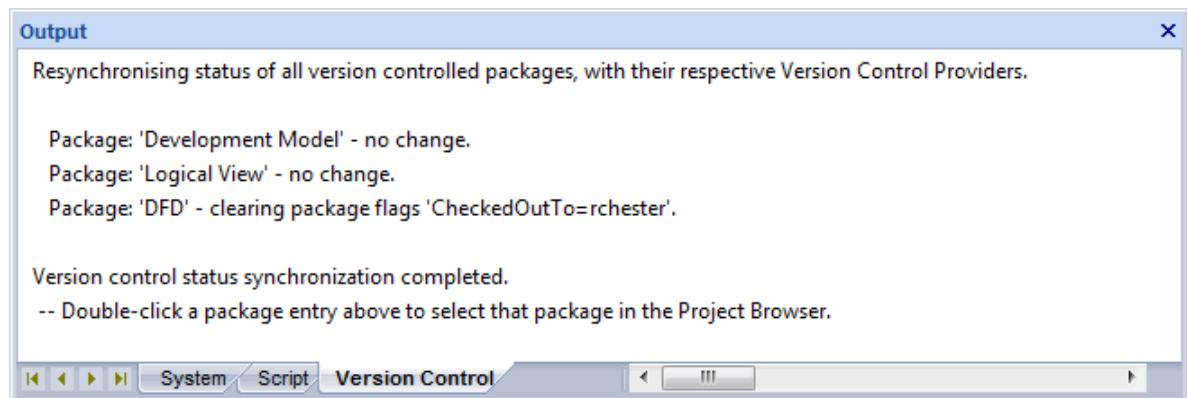
6.12 Resynchronize the Status of Version Controlled Packages

Enterprise Architect enables you to re-synchronize the version control status of either a single version controlled package or all version controlled packages within your current model with the status reported by your version control provider. This can be useful when you give a copy of a .EAP file configured for version control to a new team member.

- For a single package, right-click on the package name in the **Project Browser** and select the **Package Control | Re-synch Status With VC Provider** menu option.
- For all version controlled packages within the project, click on the **Project | Version Control | Re-synch Statuses of All Packages** menu option.

For a given package, the re-synchronization process queries the corresponding version control provider to find the status of the package file associated with the version-controlled package. If necessary, the process then updates the package flags within the model database, to synchronize the package status recorded in the model with the value reported by the version control provider.

The results of the re-synchronization process are sent to the Enterprise Architect **Output** window, as shown below:



Double-click on any result message to select, in the **Project Browser**, the corresponding package.

Note:

This process does not cause any package data to be:

- exported from your model to the associated package file, or
- imported from a package file into your model's package data.

If a package has been checked-out and modified with Enterprise Architect, but your version control provider reports the package file as checked-in, running this process marks the package within Enterprise Architect as being checked-in, without exporting and committing the pending changes. Subsequently checking-out the package imports the latest revision of the package file from version control, effectively discarding the uncommitted modifications from the model.

Similarly, if a package file is checked-out in the version control system, but not in the Enterprise Architect model, running this process marks the package within the model as checked-out, but it does not import the associated package file from the version control system. Consequently, it is possible to check-in a package from Enterprise Architect that is potentially out of date, compared to the latest revision of the package file within the version control system.

7 Offline Version Control

When loading a model that uses version control, Enterprise Architect normally initializes a connection to the version control system for each Version Control Configuration defined in the model. If Enterprise Architect is unable to connect a Version Control Configuration for any reason, it displays warning messages to notify you and provides 'offline' version control functionality for all packages associated with the failed connection.

You can prevent Enterprise Architect from attempting to make any version control connections by selecting the **Project | Version Control | Work Offline** menu option before loading a model. This is useful if you know that Enterprise Architect cannot connect to your version control system. For example, if you are working on a laptop computer that is disconnected from your network and you have an Enterprise Architect model that uses a large number of Version Control Configurations, choosing to work offline before you load the model enables you to avoid all the error messages that Enterprise Architect would normally display as each version control connection attempt fails.

You can switch between working offline and working online at any time, either before or after a model is loaded. Toggle the **Project | Version Control | Work Offline** menu option. Enterprise Architect disconnects or reconnects version control (depending on connection availability) according to your selection.

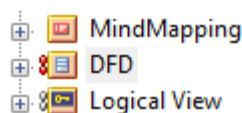
Use Version Control Whilst Disconnected From Your Version Control Server

Enterprise Architect 'remembers' the status of a model's version controlled packages. Packages that were [checked out](#)^[38] to you prior to disconnecting from the server are still shown as checked out to you, even though you are no longer connected to the server. You can still edit these packages as you normally would.

Packages that were not checked out to you prior to disconnecting from the server are shown as version controlled and locked. You cannot edit these packages until you check them out.

Offline Check Out

In releases of Enterprise Architect from release 6.0 onwards, you can 'check-out' and edit a version controlled package even when your machine is disconnected from the version control server. In the example below, the [colored 'figure 8' icon](#)^[4] for *DFD* indicates that you have checked it out whilst offline (the gray 'figure 8' icon shown against *Logical View* indicates that you have checked out a version-controlled package online).



Important:

You should be aware that the version control system - and therefore other users - have no way of knowing that you have 'checked-out' a package whilst offline. It is not possible to merge changes to an XML file that result from two users editing the same package at the same time. If an offline checkout leads to two people editing the same package at the same time, when the changes are brought back online the first-saved set of changes is lost.

Check in a Package That Was Checked Out Offline

Once you reconnect your machine to the version control server, if the package you checked out offline is not currently checked out by another user, you can check in that package. However, before Enterprise Architect checks in such a package, it compares the local working copy of the package file with the latest revision in the repository. (These package files remain unchanged in your work area until Enterprise Architect exports the package again before checking in.) If the repository version remains unchanged from when you last updated your local copy, Enterprise Architect exports and checks in your package without further prompting.

On the other hand, if the repository now contains a file that has changed since you last updated your local copy, checking in your package overwrites whatever those changes might be. Enterprise Architect displays a message warning you of the pending data loss and giving you the opportunity to abort the check in. At this point, you must decide whether to discard your own changes, using the **Undo Check Out** command, or continue with your check in and overwrite the changes that have been committed to the repository since you last updated your local copy from the repository.

You can use the **File Properties** command to determine who checked in the last changes to this package.

This might help you to discover what changes have been uploaded and decide whose changes take precedence.

Update Before You Disconnect

Whenever you are connected to the version control server, you are always working with the latest version of a package. This is because you cannot modify a package until you check it out from version control, and checking it out loads the latest revision from the repository into your model.

These rules do not apply when you are disconnected from the version control server. You are working on whatever versions you have on your machine, dating back to the last time you updated your local copy of each version controlled package. So, if you are planning to work on a model whilst disconnected from version control, it is a very good idea to make sure that you have the latest versions of all packages before you disconnect. The [Get All Latest](#)^[33] option makes this a simple task.

Index

- C -

- Check In
 - Branch 38
 - Explanation 33
 - Offline Packages 46
 - Packages Online 38
 - Project Browser Icon 38
- Check Out
 - Branch 38
 - Explanation 33
 - Packages Offline 38, 46
 - Project Browser Icon 38
- Configure
 - Package For Version Control 36
 - Version Control, Subversion 26
- Controlled Package 36
- CVS
 - Remote Repository 15
 - Version Control Options 15, 19
 - Version Control With Local Repositories 19

- E -

- EAB File
 - Export 42
 - Import 42
 - Model Branch File 42
- Export
 - .EAB File 42
 - Version Controlled Model Branch 42

- F -

- File
 - .EAB 42

- I -

- Import
 - .EAB File 42
 - Version Controlled Model Branch 42

- M -

- Menu

- Version Control 33
- Model
 - Connect To Version Control Using TFS 29
- Model Branch
 - .EAB File 42
 - Apply Version Control 41
 - Check In 38
 - Check Out 38
 - Export 42
 - File 42
 - Import 42

- N -

- Nested Version Control Packages 6, 11

- O -

- Offline
 - Checkout 46
 - Version Control 46

- P -

- Package
 - Apply Version Control To Model Branch 41
 - Check In 38
 - Check Out 38
 - Configure For Version Control 36
 - Controlled 36
 - Export Version Controlled Model Branch 42
 - Import Version Controlled Model Branch 42
 - Nested in Version Control 11
 - Resynchronize Package Version Control Status 45
 - Review Version Control History 44
 - Validate Version Control Configuration 37
 - Version Control 2
- Private Model
 - And Version Control Using TFS 29
- Project Branch
 - Check In 38
 - Check Out 38
- Project Browser
 - Version Control Indicators 2

- R -

- Refresh
 - Diagram 44

Refresh
 Project 44
 View Of Shared Model 44

Reload
 Diagram 44
 Model (Shared) 44
 Project 44
 View 44

Resynchronize
 Package Version Control Status 45

Review
 Package Version Control History 44

- S -

SCC
 Providers Dialog 11
 Version Control Options 11
 Version Control, Upgrade For Enterprise Architect 4.5 14

Source Code
 Control 2

Subversion
 Caching Client Credentials 22
 Configure Version Control 26
 Documentation 22
 Executables 22
 Repository URLs 22
 Setting Up 22
 TortoiseSVN 28
 UNIX-Based Client 24
 Using With Enterprise Architect Under WINE Crossover 24
 Version Control Options 22
 Version Control, Create Local Working Copy 24
 Version Control, Create Repository Subtree 23
 Windows-Based Client 24

- T -

Team Deployment
 And Version Control 7
 Version Control Branching 7

Team Foundation Server
 Connect Enterprise Architect Model For Version Control 29
 Version Control Option 28

TFS
 Connect Enterprise Architect Model For Version Control 29
 Version Control Option 28

Working Folder For Enterprise Architect Version Control 28
 Workspace For Enterprise Architect Version Control 28

TortoiseSVN
 In Version Control 28

- V -

Validate
 Package Configuration For Version Control 37

Version Control
 Apply To Enterprise Architect Model 6
 Apply To Model Branch 41
 Basics 5
 Branching 7
 Check In and Check Out Packages 38
 Configuration 9, 42
 Configuration, Team Deployment 7
 Configuration, Use Previously-Defined 37
 Configure Current Package 8
 Configure In Subversion 26
 Configure Package 36
 Connect Model Using TFS 29
 Copy-Modify-Merge Policy 5
 CVS Options 15, 19
 CVS Remote Repository 15
 Discussion Of File Control 6
 Export Model Branch 42
 File History 33
 File Properties 33
 Import Model Branch 42
 In Team Deployment 7
 Include Other Users' Packages 40
 Introduction 2
 Locking - Necessary? 5
 Lock-Modify-Unlock Policy 5
 Menu 33
 Nested Packages 6
 Offline 46
 Package Configuration, Validate 37
 Policies 5
 Project Browser Indicators 2
 Recommendations 33
 Refresh View Of Shared Model 44
 Resynchronize Package Version Control Status 45
 Resynchronize Status Of All Packages 8
 Review Version Control History 44
 SCC Options 11
 SCC, Providers Dialog SCC, Providers Dialog 11

Version Control

- SCC, Upgrade For Enterprise Architect 4.5 14
- Set Up 9
- Settings 8, 9
- Setup Menu 8
- Subversion Options 22
- Subversion, Create Local Working Copy 24
- Subversion, Create New Repository Subtree 23
- Subversion, Setting Up 22
- Subversion, TortoiseSVN 28
- Subversion, Using With Enterprise Architect Under WINE Crossover 24
- Tasks 33
- TFS Option 28
- Use Nested Version Control Packages 11
- Using 33
- Validate Package Configurations 8
- Who Has Checked Out A Package? 33
- Work Offline 8

- W -

WINE-Crossover

- Using Subversion With Enterprise Architect Under 24

Version Control Within UML Models
Using Enterprise Architect

www.sparxsystems.com