



Embarcadero Describe

Working with Configuration Management Applications

by
Brent Hansen
Product Manager

Jeff Bush
Software Consultant

9/12/2003

**Copyright © 1994-2003 Embarcadero Technologies, Inc.
San Francisco**

Copyright © 1994-2003 Embarcadero Technologies, Inc.

All rights reserved.

Embarcadero Technologies, Inc.

425 Market Street, Suite 425

San Francisco, CA 94105

ER/Studio is a trademark of Embarcadero Technologies, Inc.

All other brand and product names are trademarks or registered trademarks of their respective owners.

This software/documentation contains proprietary information of Embarcadero Technologies, Inc.; it is provided under a license agreement containing restrictions on use and disclosure and is also protected by copyright law. Reverse engineering of the software is prohibited.

If this software/documentation is delivered to a U.S. Government Agency of the Department of Defense, then it is delivered with Restricted Rights and the following legend is applicable:

Restricted Rights Legend Use, duplication, or disclosure by the Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of DFARS 252.227-7013, Rights in Technical Data and Computer Software (October 1988).

If this software/documentation is delivered to a U.S. Government Agency not within the Department of Defense, then it is delivered with Restricted Rights, as defined in FAR 552.227-14, Rights in Data-General, including Alternate III (June 1987).

Information in this document is subject to change without notice. Revisions may be issued to advise of such changes and additions. Embarcadero Technologies, Inc. does not warrant that this documentation is error-free.

Table of Contents

Introduction to Describe's Configuration Management Framework.....	4
Understanding Describe's Default Support for Configuration Management	5
Default Configuration Management Preferences.....	6
Default Configuration Management Applications.....	10
Default Configuration Management Detection	10
Switching Between SCC Providers.....	10
Custom Configuration Management Applications.....	10
Default Versionable Elements	11
Versioning Source Code.....	12
Describe Architecture Using XMI.....	15
Workspaces.....	15
Creating Workspaces	16
Projects	17
Creating Projects	18
Diagrams	19
Packages	19
Elements	19
Understanding XMI Protocol for Describe Architecture and Versioning.....	20
Understanding XMI Architecture.....	20
Importance of Versioning Packages.....	23
Configuration Management Commands inside of Describe	24
Source Code Control.....	24
Working with the Workspace View Pane.....	24
Available Configuration Management Commands	25
Get Latest Version	25
Get Latest Including All Scoped Elements.....	26
Check In	26
Check Out.....	27
Show History.....	27
Show Differences.....	28
Add to Source Control	28
Remove from Source Control	29
Launch Provider.....	29
Refresh Status	29
Open a Workspace from CM Tool.....	29
Tips and Tricks while working with Configuration Management using Describe	30
Best Practice Scenarios	30
Versioning at the Package Level Example	30
Sharing Elements across Packages Example	31
Dividing Large Projects	32
References Libraries.....	33
Extended Versionable Elements for Power Users.....	34
Extended Versionable Element.....	37

Introduction to Describe's Configuration Management Framework

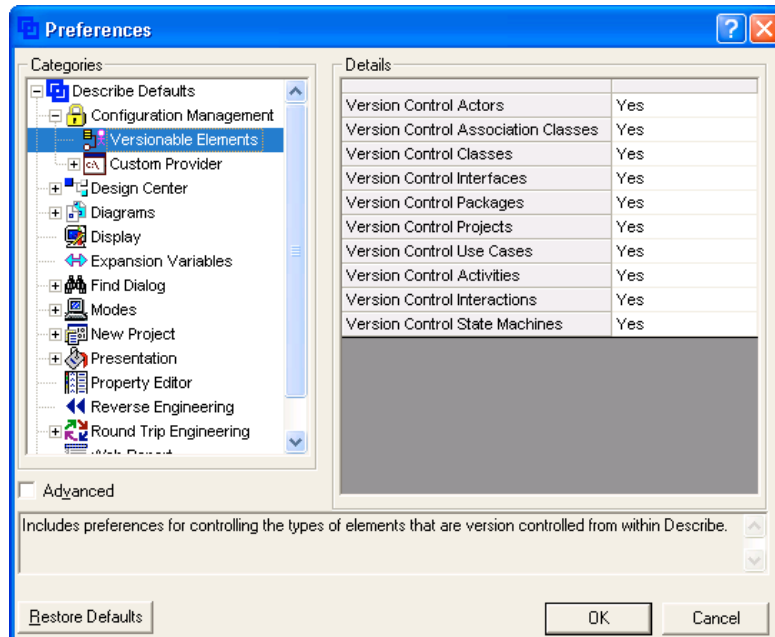
Configuration Management (CM) enables you to baseline, rollback and set releases from historical and present information. It is the process of entering various artifacts of the software (or business) life cycle into a central repository for sharing and versioning purposes across the Enterprise.

Describe integrates with Source Code Control Interface (SCCI)-compliant Configuration Management applications as well as Concurrent Versions Systems (CVS). Describe utilizes various configuration management providers for versioning source code, UML meta data, or any other artifact that can be associated with modeling artifacts, directly from its own context menus. The granularity of versioned artifacts is completely user-defined, allowing you to version everything from the workspace down to an individual class.

Describe stores data in a variant XMI format, an XML representation of the UML Meta data. We say "variant" in that the XMI 2.0 standard was not ready when the Describe development team designed the persistence layer, so the XMI 1.4 format was enhanced with UML 2.0 constructs, as well as Describe-specific internal Meta data. This file-based method of storage gives Describe the ability to support the versioning of elements on a granular level within existing CM repositories. Individual class elements, as well as workspaces and projects can be versioned, allowing team members to simultaneously work on these elements.

Understanding Describe's Default Support for Configuration Management

Describe lets you manage version control and collaboration by offering configuration management functionality through context menus in the Workspace View pane. By default, you can version control Describe workspaces and any Meta type located in the Preferences dialog box under **Describe Defaults>Configuration Management>Versionable Elements**.



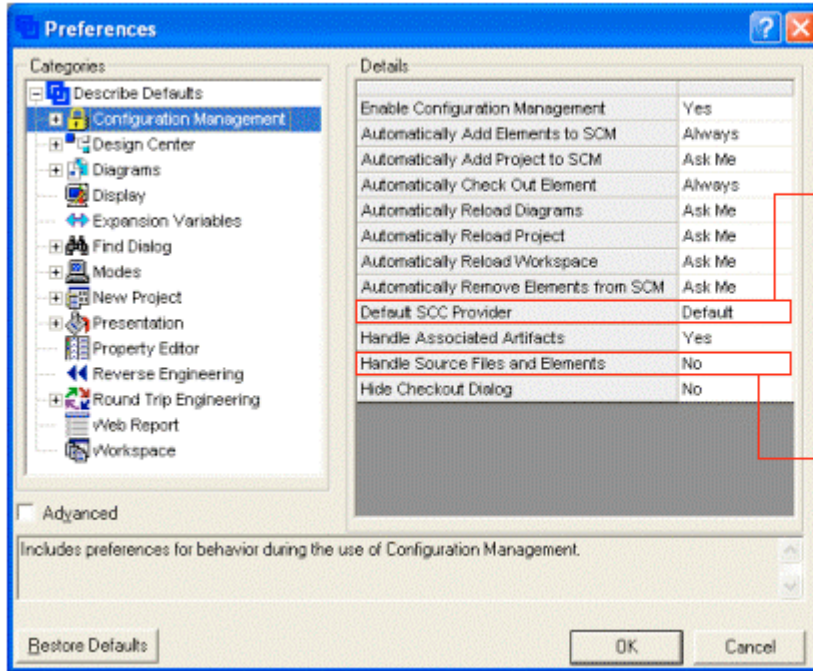
Before working with the various configuration management options within Describe, you need to determine your workflow for collaboration and versioning granularity for models and decide whether to version source code using Describe.

Upon first launch of the Describe application (either stand-alone or via the various IDMEs supported by Describe), Describe is configured to work with the current / default Source Code Control (SCC) provider. These providers include the major configuration management tools, such as IBM's ClearCase, Microsoft's Visual SourceSafe, Merant's PVCS Version Manager and Perforce.

Using the Preferences dialog, you can choose whether to use the custom command line provider or the default SCC provider.

- Set the Default SCC Provider preference to **Default** if using a SCCI-Compliant CM Tool.
- Set the Default SCC Provider preference to **Custom** if using CVS.

This is where you define the property setting to version source code along with modeling elements (classes) via a configuration management tool.



Default Configuration Management Preferences

Describe enables you to modify the default Configuration Management settings via the Preferences dialog. To view Describe preferences, select Preferences from the Edit menu. The Configuration Management settings are located under the Configuration Management node.

The Source Configuration Management (SCM) Preferences are below:

Detail	Explanation
Enable Configuration Management	Determines whether or not configuration management is enabled when using Describe.

<p>Automatically Add Elements to SCM</p>	<p>Determines whether or not elements that are added to a versioned project are automatically added to source control as they are named.</p> <p>Yes: Elements are automatically added to source control as they are named.</p> <p>No: Elements are not automatically added to source control.</p> <p>Ask Me: You are prompted to make a decision each time an element is named.</p>
<p>Automatically Add Project to SCM</p>	<p>Do you want to be prompted to version control the owning project of any elements you are versioning?</p> <p>Always: The project is automatically added to source control.</p> <p>Never: Neither the project nor the element is added to source control.</p> <p>Ask Me: You are prompted to make a decision each time you add an element to source control whose owning project has not already been added to source control.</p>
<p>Automatically Check Out Element</p>	<p>Determines whether or not you are prompted to check out an element from source control if it has been versioned and you attempt to make modifications to the element.</p> <p>Always: Elements are always automatically checked out.</p> <p>Never: Elements are never automatically checked out.</p> <p>Ask Me: You are prompted to make a decision each time you attempt to modify a versioned element.</p>

<p>Automatically Reload Diagram</p>	<p>Do you want diagrams automatically reloaded when checking out from source control, undoing a checkout, or getting the latest version?</p> <p>Always: Diagrams are automatically reloaded.</p> <p>No: Diagrams are never reloaded.</p> <p>Ask Me: You are prompted to make a decision each time a diagram needs to be reloaded.</p>
<p>Automatically Reload Project</p>	<p>Determines whether or not projects are automatically reloaded when checking out a project from source control, undoing a checkout, or getting the latest version of a project.</p> <p>Always: Projects are automatically reloaded.</p> <p>No: Projects are never reloaded.</p> <p>Ask Me: You are prompted to make a decision each time a project needs to be reloaded.</p>
<p>Automatically Reload Workspaces</p>	<p>Determines whether or not workspaces are automatically reloaded when checking out a workspace from source control, undoing a checkout, or getting the latest version of a workspace.</p> <p>Always: Workspaces are automatically reloaded.</p> <p>No: Workspaces are never reloaded.</p> <p>Ask Me: You are prompted to make a decision each time a workspace needs to be reloaded.</p>
<p>Automatically Remove Elements from SCM</p>	<p>Determines whether or not elements are automatically removed from source control when they are deleted from a versioned project.</p> <p>Yes: Elements are automatically removed from source control as they are deleted.</p> <p>No: Elements are not automatically removed from source control.</p> <p>Ask Me: You are prompted to make a decision each time an element is deleted from a project.</p>

Default SCC Provider	<p>Determines the Source Code Control provider to be used within Describe.</p> <p>Default: The default provider is used.</p> <p>Custom: The commands for performing various source control actions must be set under the Custom Provider node.</p>
Handle Associated Artifacts	<p>Determines whether or not associated artifacts are handled along with the elements owning those artifacts when working with source control.</p> <p>Yes: Artifacts are handled with their associated elements when adding to source control, checking in, checking out, etc.</p> <p>No: Only the element itself is handled. Any handling of artifacts must be done manually outside of Describe.</p>
Handle Source Files and Elements	<p>Determines whether or not source files are handled along with their associated elements when working with source control.</p> <p>Yes: Source files are handled with their associated elements when adding to source control, checking in, checking out, etc.</p> <p>No: Only the element itself is handled. Any handling of source files has to be done manually outside of Describe.</p>
Hide Checkout Dialog	<p>Determines whether or not the Configuration Management dialog is hidden when checking elements out from source control.</p> <p>Yes: The dialog is hidden.</p> <p>No: The dialog is displayed. When this preference is set to No, you can still cause the dialog to be displayed by pressing SHIFT while checking out elements.</p>

Default Configuration Management Applications

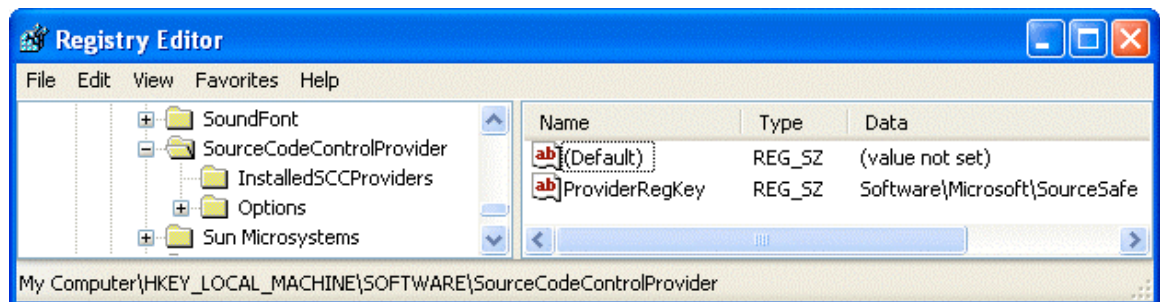
Describe integrates with SCCI-compliant Configuration Management applications, such as:

- Borland Star Team (www.borland.com)
- IBM ClearCase (www.ibm.com)
- Merant PVCS (www.merant.com)
- Microsoft Visual Source Safe (www.microsoft.com)
- Perforce (www.perforce.com)

Default Configuration Management Detection

Describe detects the Default Configuration Management Provider from the Windows Registry. You do not have to make any modifications in order for Describe to seamlessly integrate with an SCC provider.

Describe uses the **HKEY_LOCAL_MACHINE\SOFTWARE\SOURCECODECONTROLPROVIDER\ProviderRegKey** key, as illustrated below.



Switching Between SCC Providers

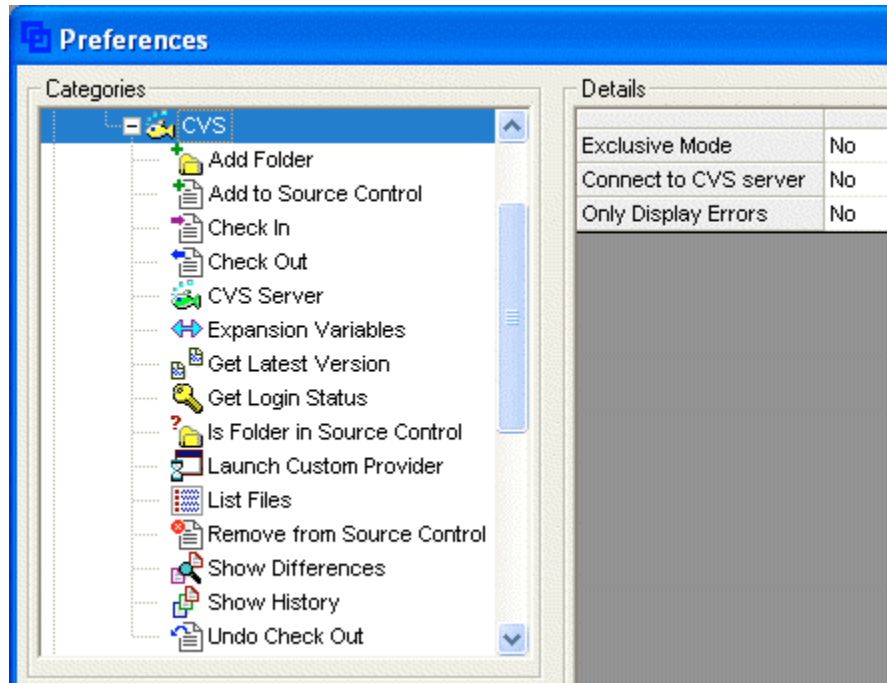
Describe enables you to switch between SCC providers by changing the “ProviderRegKey” value or installing a new SCC provider in which the SCC Provider Installation updates the registry.

Name	Type	Data
(Default)	REG_SZ	(value not set)
ProviderRegKey	REG_SZ	Change the value of this key to use a different SCC provider.

Custom Configuration Management Applications

Describe can also work with Configuration Management tools that do not support the SCC interfaces through additional preference settings. One common tool that falls into this category is the Concurrent Versioning System (CVS), an open

source CM tool. Describe includes a predefined set of command line commands that enables Describe to interface with CVS out of the box.



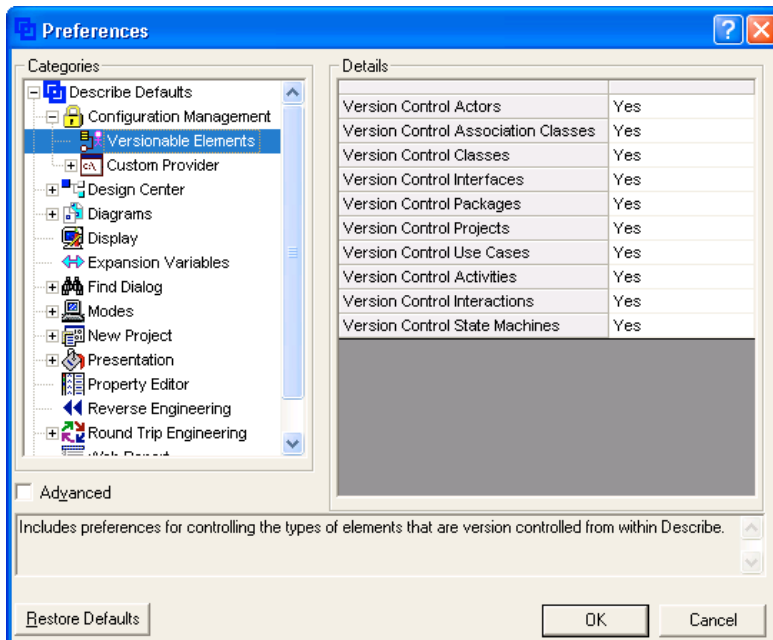
For details on using the CVS integration with Describe, refer to the following White Paper: <http://www.embarcadero.com/products/describe/demanuals.asp>

Default Versionable Elements

Describe enables you to version the following Meta data:

- Workspace
- Projects
- Diagrams
- Modeling Elements
 - Actors
 - Association Classes
 - Classes
 - Interfaces
 - Packages
 - Projects
 - Use Cases
 - Activities
 - Interactions
 - State Machines

In the Preferences dialog, under the **Configuration Management>Versionable Elements** node, you can decide whether or not each element can be version controlled from within Describe. When set to Yes, elements can be version controlled. When set to No, they cannot be version controlled from within Describe.



NOTE: Describe allows Power Users to modify XML configuration files to version any modeling element (such as a parameter). You can also version artifacts, including scripts or documentation pertaining to your project. Refer to [Extended Versionable Elements for Power Users](#).

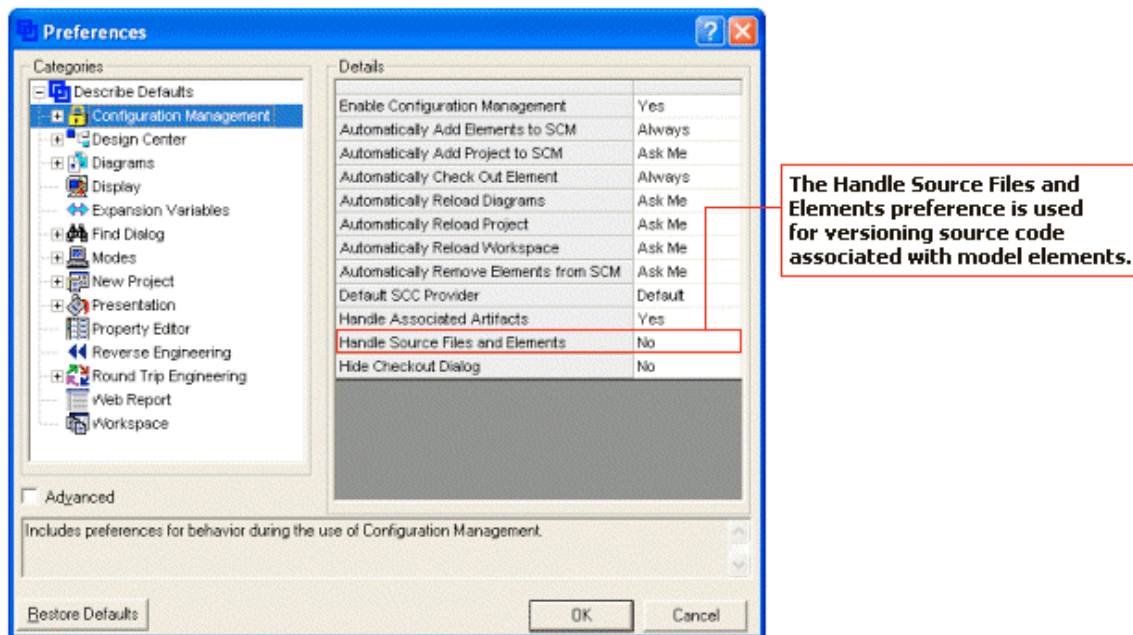
Versioning Source Code

Describe enables you to version the source code created with the following features:

- Reverse Engineering
- Code Generation
- Live Round Trip

Describe, by default, sets Source Code versioning ability to No. You can change this option to Yes in order for Describe to version the source code file artifacts along with corresponding modeling elements (classes, interfaces, etc.)

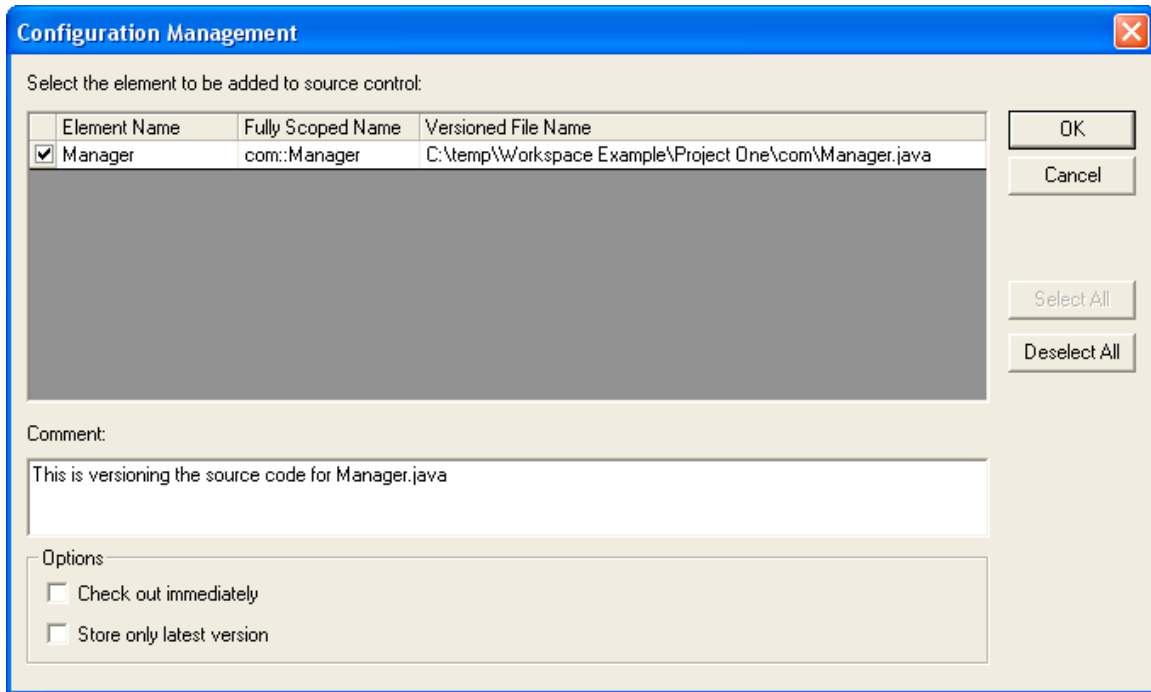
The illustration below shows that the Handle Source Files preference is set to No. This command does not prompt you to version source as it is generated.



For example, when working in Implementation Mode (Live Round Trip), Describe prompts you to version the source code file artifact when this preference is set to Yes.

NOTE: This applies to all source code languages supported by Describe.

The following dialog displays the source code artifact being versioned through the Describe interface via the Configuration Management provider integration. You can check the file out immediately, store only the latest version, and comment the check in.



Describe Architecture Using XMI

Describe uses XMI Meta data for storing all workspaces, projects, packages, diagrams, and modeling elements. Because XMI is simply an XML representation of UML Meta data represented in a simple ASCII text file, you can version control elements at a much more granular level.

Workspaces

A workspace is a unit of storage used for any collection of data modeling projects in Describe. You can only open and work with one workspace at a time, but you can collect any number of projects in a single workspace. Describe lets you create, name, save, reopen, and close workspaces.

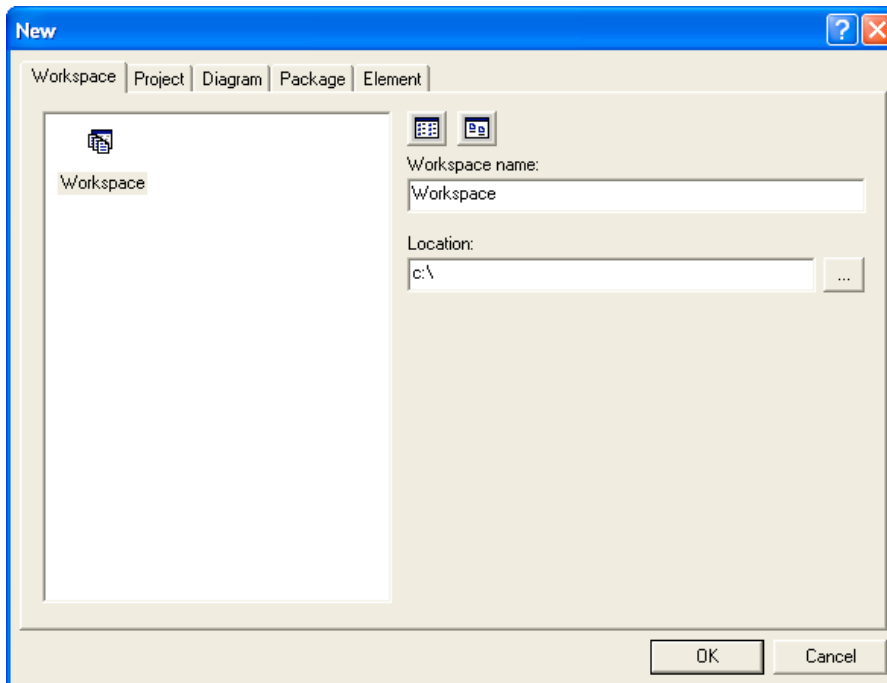
- In the Workspace View pane, your workspace occupies the root node under which all projects and their associated elements appear in a tree format.
- On creation, Describe lets you specify the name and location of your workspace and saves it as a *.etw file.

Creating Workspaces

Describe allows you to create Workspaces in the following ways:

- From the main Describe menu, select **File>New**.
- Use the keyboard shortcut **CTRL+N**.
- Right-click in the Workspace View pane and select **New>Workspace**.

The **New** dialog box is displayed:



NOTE: See the General Tutorial for more information.

Once you have created the initial workspace, you have several options for starting a project. Click [here](#) for more information on starting projects.

1. You are asked to create a new project based on the creation of a workspace.
2. Right-click the workspace node in the Workspace View pane and select **Source Control>Add to Source Control**.
3. The workspace acts as a working container for newly created and imported projects.

Projects

A project is a unit of storage for any collection of diagrams and their associated elements in Describe. The application lets you organize your projects into distinctly different workspaces.

- You can move projects between workspaces using the insert project feature.
- The Workspace View pane displays each project node under a single root workspace node. You can expand each project in the Workspace View pane to display all associated elements in a tree format.
- On creation, Describe lets you specify the name and location of your project, and saves it as a *.etd file in a folder located on the same level as the new project's workspace.
- You can open and work with multiple projects at a time.

Projects can be added to a workspace by:

- Creating a new one.
- Importing an existing project.
- Reverse Engineering source code into new projects.
- Creating a new project via the ER/Studio integration.
- Importing from a Rose Model.
- Upgrading Describe 5.x systems.

NOTE: Describe creates a corresponding workspace, project, and package directory structure (working structure). The integrity of this working structure must match the sandbox, view, or project directory structure within the CM tool. This is specific to each CM tool. Specifically, if you want to use the [Open Workspace from SCM](#) feature, the workspace and associated projects need to share a common root in the repository.

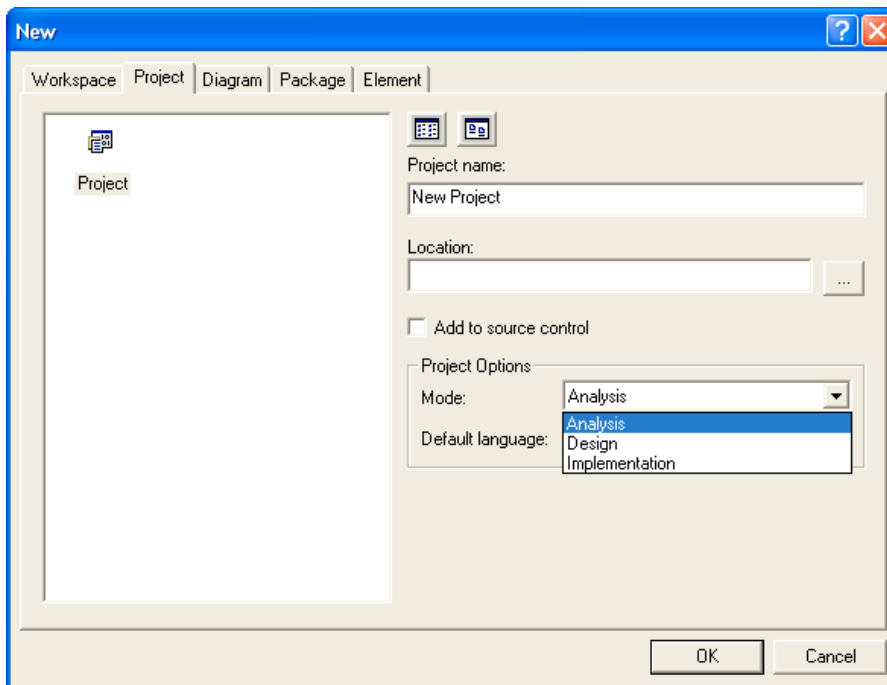
Creating Projects

You can create a new project by selecting **File>New** from the main Describe menu.

Project Modes

You have three options when setting the Project Mode:

- Analysis
 - Pure modeling with no language-specific restrictions placed on Meta type creation.
 - No code is automatically generated (Round Trip Engineering is off).
- Design
 - Round Trip Engineering is off.
 - Language-specific constraints are placed on the models (e.g. if the project's Language property is set to "Java", no multiple inheritance generalization link relationships are allowed).
- Implementation
 - Round Trip Engineering is on.
 - Language constraints are on.



You have the ability to add this newly created project directly into source code control. Once the project has been added to the CM Tool, it is available for check out. Before going further into modeling, you need to understand how Describe treats versioned projects, packages, and design elements.

Diagrams

A diagram is a graphical presentation of a collection of model elements often rendered as connected graphs of arcs and vertices. UML supports the following diagrams: class, object, use case, sequence, collaboration, state, activity, component, and deployment.

Packages

A package is a general purpose mechanism for organizing elements into groups. Packages may be nested within other packages.

Elements

An Element is a UML Notation representation of Meta data.

Understanding XMI Protocol for Describe Architecture and Versioning

Because of the way Describe stores elements, you can now version individual class elements as well as workspaces and projects, allowing team members to simultaneously work on these elements. For example, team members can simultaneously work on the same class diagram, with one member working on the class diagram itself while the other member works on the classes within that class diagram.

Using XMI architecture allows Describe to take advantage of Dynamic Loading technologies. This allows you to only load into memory the Meta data you are viewing or manipulating. Hence, performance is maximized across the workspace and projects.

Understanding XMI Architecture

You need a clear understanding of how Describe stores information and the XMI architecture in order to be successful in collaboration using Describe.

For organizational purposes, it is recommended that the best practice for working in collaboration mode would be to designate a Project Lead. The Project Lead would be responsible for creating workspaces, projects, and package structure.

Once these have been established you need a deep understanding of the XMI protocol in how elements are stored.

- Workspace
 - The workspace is a sandbox for projects.
 - The workspace XMI file extension is *.etw.
 - The *.etw file contains tagged values that reference projects.

```
<EMBT:WSProject name="ProjectOne"
baseDirectory=".\\ProjectOne" scmID="Custom"><EMBT:WSElement
href="./ProjectOne/ProjectOne.etd">
```

NOTE: The workspace node only needs be versioned when adding new or removing projects. The workspace also needs to be versioned when adding documentation or artifacts at the workspace level.

- Project
 - The project is an organized container for all the diagrams and modeling elements.

- The project XMI file extension is *.etd.
 - This is an example of an *.etd snippet with a package that is NOT versioned.


```
<UML:Package xmi.id="DCE.D5AA406A-4BA6-432D-B2DA-09AD0ACA16F6" owner="DCE.E718B476-95AD-48AD-B87D-06B302A8BF86" name="designPackage"/>
```
 - This is an example of an *.etd snippet with a versioned package.


```
<UML:Package xmi.id="DCE.C00048A1-8817-4330-9632-D9DB381A1200" owner="DCE.F060269B-811C-4062-BCB6-DE758F65D20E" name="designPackage" isVersioned="true" isDirty="false" href=".\\Ver103.etx#/*[@xmi.id=&quot;DCE.C00048A1-8817-4330-9632-D9DB381A1200&quot;]"/>
```
 - The highlighted portion of the XML above dictates a versioned element. NOTE: **HREF** points to a unique versioned file.
 - In order to eliminate CM Tool merging conflicts, Project Leaders and Developers should work in their own respective packages, similar to working with production source code.
 - If the Development team is only working in one package or no packages, elements and diagrams should be versioned individually.
- Diagrams
 - The diagram presentation data is stored as *.etlp.
 - The diagram layout information is stored as *.etld.
 - Diagrams are saved in the following locations:
 - Diagrams scoped to the project level are stored within the project directory.
 - Diagrams scoped to the package level are stored within the package directory.

- Versioned Element
 - A version element has the XML file extension *.etx
 - The *.etx file is auto generated for each versioned element with a random name.
 - For example, you version class “Customer”.
 - Ver103.etx file is created, which represents a random string file name for versioned class “Customer”.

NOTE: The two examples below demonstrate a scenario of how Describe handles XML tagged values between versioned files and non-versioned files (only saved modeling elements).

The following is an example of a versioned package with two non-versioned classes named “Manager” and “Customer”:

```
<UML:Element.ownedElement>
<UML:Class xmi.id="DCE.902C34BB-5062-41F6-A260-2C21D58A78A2"
owner="__uri_.\Ver103.etx#/*[@xmi.id=&quot;DCE.C00048A1-8817-4330-9632-
D9DB381A1200&quot;]" name="Manager">
<UML:Element.presentation>
</UML:Element.presentation></UML:Class>
<UML:Class xmi.id="DCE.C9799E41-93A5-430C-8AF9-7055837E0D30"
owner="__uri_.\Ver103.etx#/*[@xmi.id=&quot;DCE.C00048A1-8817-4330-9632-
D9DB381A1200&quot;]" name="Customer">
<UML:Element.presentation></UML:Element.presentation></UML:Class></UML:Element.owned
Element>
```

The following is an example of a Versioned Package with two versioned classes named “Manager” and “Customer”:

```
<UML:Element.ownedElement>
<UML:Class xmi.id="DCE.902C34BB-5062-41F6-A260-2C21D58A78A2"
owner="__uri_.\Ver103.etx#/*[@xmi.id=&quot;DCE.C00048A1-8817-4330-9632-
D9DB381A1200&quot;]" name="Manager" isVersioned="true" isDirty="false"
href=".Ver10D.etx#/*[@xmi.id=&quot;DCE.902C34BB-5062-41F6-A260-
2C21D58A78A2&quot;]">
</UML:Class>
<UML:Class xmi.id="DCE.C9799E41-93A5-430C-8AF9-7055837E0D30"
owner="__uri_.\Ver103.etx#/*[@xmi.id=&quot;DCE.C00048A1-8817-4330-9632-
D9DB381A1200&quot;]" name="Customer" isVersioned="true" isDirty="false"
href=".Ver10C.etx#/*[@xmi.id=&quot;DCE.C9799E41-93A5-430C-8AF9-
7055837E0D30&quot;]">
</UML:Class></UML:Element.ownedElement>
</UML:Package></VersionedElement>
```

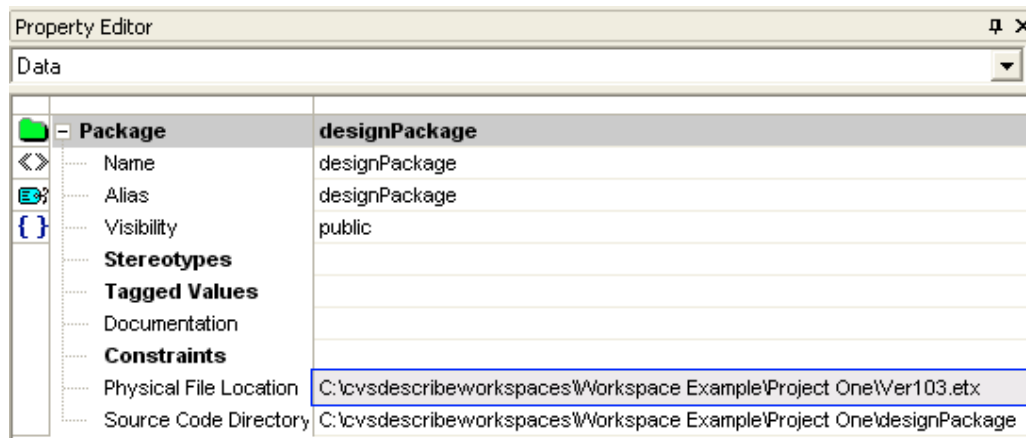
NOTE: [href=".Ver10D.etx#//](#) references a versioned element in the example class “Manager”.

Importance of Versioning Packages

Why is it important that you version at the package and element level?

Once a modeling element is versioned, it creates a corresponding XMI file. The architecture allows you to check out these versioned files at a granular level without locking all the owner elements (i.e. packages and projects). Hence, separate developers are able to work in a collaborative mode sharing the project without creating merging conflicts. Furthermore, this intended workflow is the most efficient collaborative environment. Describe uses XPointer technology to quickly load and reference desired information.

You have the ability to view the path and versioned file name from the Property Editor under Physical File Location.



Configuration Management Commands inside of Describe

Describe includes support for SCCI-compliant providers and customer provider interfaces, which lets you drive your Configuration Management provider from the Describe GUI to version Describe elements. The integrated source code control also lets you collaborate with others in the creation and modification of Describe elements with little cost to your network performance.

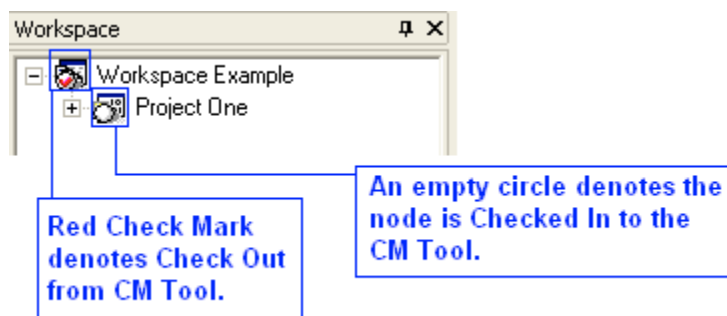
Source Code Control

Describe supports SCCI-compliant SCM tools and CVS. Describe stores elements as XMI, an XML representation of the UML Meta data. This method of storage gives Describe the ability to support the versioning of elements on a granular level. You can now version individual class elements as well as workspaces and projects and team members can simultaneously work on these elements. For example, team members can simultaneously work on the same class diagram, with one member working on the class diagram itself while the other member works on the classes within that class diagram.

You can also version artifacts, including scripts or documentation pertaining to your project.

Working with the Workspace View Pane

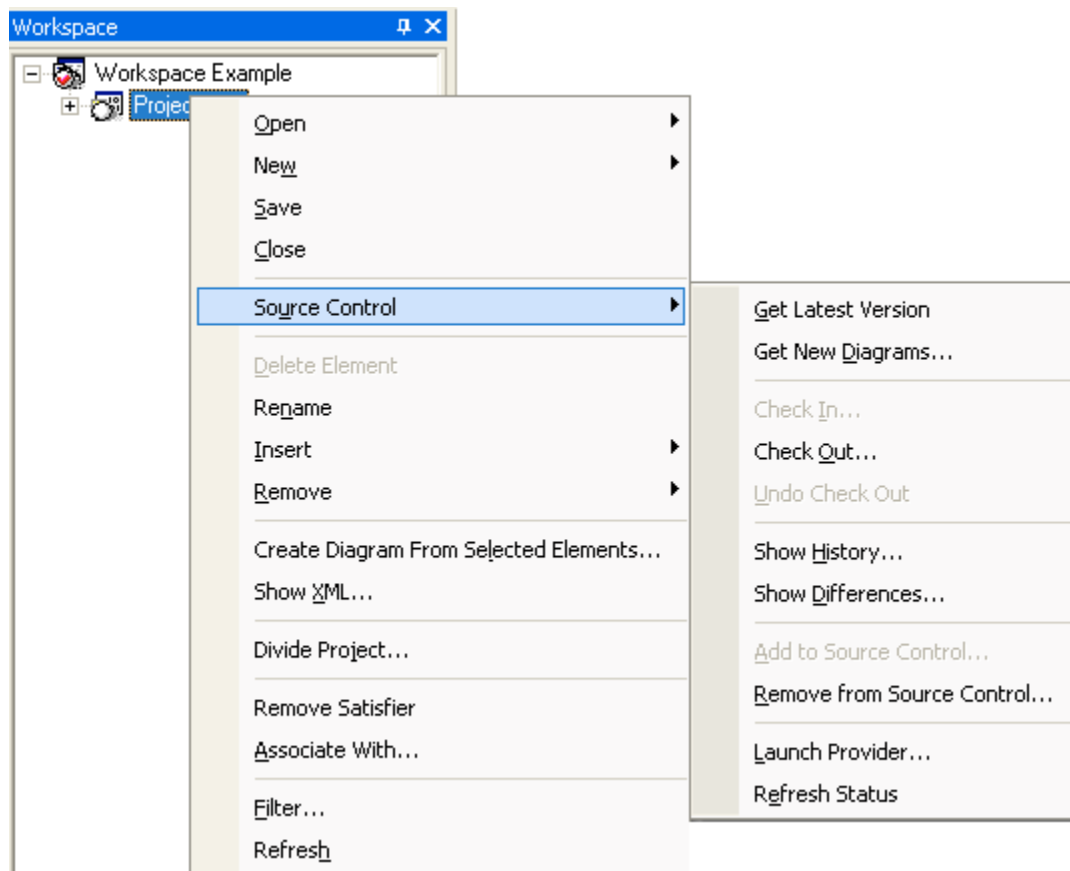
Describe alters the appearance of certain icons in the Workspace View pane, depending on their state of source control. The icon alterations that occur when working with source control is described as follows:



Available Configuration Management Commands

You can view the Configuration Management Commands by selecting the node from the Workspace View pane

- Right-click and select Source Control.



Get Latest Version

Describe lets you get the latest version of any Describe element added to Configuration Management. Describe then updates that element in the Describe Workspace with the last checked-in version of that file. If you are working in a collaborative environment, this is a good way to see the most recent changes made by other team members.

NOTE: To get updated diagrams from other users, you must open your CM provider and do a manual **Get New Diagrams**. This updates the Workspace View pane with latest changes.

If you are working on a class diagram and another member of your team is working on a class in that class diagram, you can get the latest version of the class to see the changes made by your team member. This element is

designated as read-only. If you want to edit an element that is under source code control, you must check out the element.

Get Latest Including All Scoped Elements

To get the latest version of an element and all related elements from source code control, use the **Get Latest Including All Scoped Elements** command on the Source Control menu.

Use this command when you want to see the current state of an element and the elements that have a relationship with that element, but do not need to edit any of the elements. If you are working in a collaborative environment, getting the latest version of an element with all of the scoped elements is a good way to see the most recent changes to an element and its context. For example, if you are working on a class and other members of your team are working on the class diagram and package that contains that class, you can see the most recent versions of those elements by getting the latest version of the class diagram with all of the scoped elements. You need not get the latest version of each associated element individually.

Describe shows the results of the **Get Latest Including All Scoped Elements** command in the SCM Operation Results window.

This element is designated as read-only. If you want to edit an element that is under source code you must check out the element.

Check In

To check an element in and commit your changes to the repository, use the **Check In** command on the Source Control menu.

You can check an element in if you want to commit changes that you make to that element to the SCC repository. When you check a file into SCC, that file's project and project elements remain editable in Describe . But if you continue to edit a file that is not checked out, you might encounter synchronization problems.

If you are working in a collaborative environment, it is good to frequently check in your changes to let your team members view the current state of the element. Describe shows the results of the **Check In** command in the SCM Operation Results window. You must add an element before checking it in. Adding an element is effectively the same as checking an element in for the first time.

If you are using CVS as your SCC provider, checking an element in is the same as committing your changes. CVS will then show the element checked in via the Workspace view pane.

You can select the Keep Checked Out option to keep the element open for further work.

Check Out

To check an element out from the repository, use the **Check Out** command on the Source Control menu.

Check an element out if you intend to make changes to an element under source code control. Describe shows the results of the **Check Out** command in the SCM Operation Results window.

Undo Check Out

You can undo the check out of an element if you do not want to commit the changes that you make to an element that you have checked out. Undoing the check out returns the element to the state it was in before you checked it out.

When you undo a check out, the SCC provider detects whether the checked out file has changed since being checked out. If you have not changed the file, Describe instructs the SCC provider to check the unaltered file back in. However, if you have changed the file, Describe prompts you with a message box asking if you want to undo the check out and lose your changes.

If you are using a custom SCC provider, Describe shows the results of the **Undo Check Out** command in the SCM Operation Results window.

Show History

To show the history of an element, use the **Show History** command from the Source Control menu.

For any element file you add to SCC, Describe lets you display that file's history using the **Show History** feature.

You can show the history of an element if you want to see the number of revisions for an element, when an element was revised, or who revised an element.

Showing the history of an element is helpful when you need to take corrective action on an element to which erroneous changes have been made.

If you use a Custom SCC provider, Describe shows the results of the **Show History** command in the Command Output Window. If you use a default SCC provider, Describe opens the appropriate dialog box in that provider. For example, if you use Microsoft Visual SourceSafe, Describe opens the History Options dialog box so that you can enter history criteria and then opens the History dialog box to display the results.

Show Differences

To show the differences between versions of an element, use the **Show Differences** command on the Source Control menu.

For any project or project element file you add to SCC, Describe lets you show the differences between any two versions of that file using the **Show Differences** feature.

You can show the differences for an element when you want to see the changes and revisions made to that element. Showing the differences is helpful to see how much has changed in an element or when a Describe element was changed. It is also helpful when you need to take corrective action on an element to which erroneous changes have been made.

If you use a Custom SCC provider, Describe shows the results of the **Show Differences** command in the Command Output Window. If you use the default SCC provider, Describe opens the appropriate dialog box in that provider. For example, if you use Microsoft Visual SourceSafe, Describe opens the Difference Options dialog box.

Add to Source Control

To add an element to source control, use the **Add to Source Control** command on the Source Control menu.

Describe lets you add any element to Configuration Management. When you create an element in Describe, the element is not automatically added to your Configuration Management provider's repository. You must add the element manually. However, Describe facilitates this process by prompting you to add an element to source code control the first time that you save an element.

When working with a project that has not been added to source control, you must complete the Configuration Management dialog box, which lets you select the elements to be added to Configuration Management, add comments to those elements, and specify whether you want to do the following:

- Check the elements out immediately.
- Store the element's latest version only.
- Add comments, if desired.

If you are using a custom Configuration Management provider, Describe shows the results of the **Add to Source Control** command in the SCM Operation Results window.

Describe automatically prompts you to add parent elements to source code control. For example, if you add a Class to source code control and the Class

Diagram that contains that Class is not already under source code control, Describe detects this discrepancy and prompts you to add the Class Diagram.

Remove from Source Control

To remove an element to source control, use the **Remove from Source Control** command on the Source Control menu.

Describe lets you remove elements from SCC. If you want to delete an element that is under source code control, you need to first remove that element from the SCC provider's repository.

If you are using a custom provider, Describe shows the results of the **Remove from Source Control** command in the SCM Operation Results window.

Launch Provider

Describe lets you start your SCC provider from within Describe.

You may want to start your SCC provider to see elements as they appear in the provider itself or to make sure that a transaction completed correctly.

In the **Workspace View** pane, right-click any element, and, under **Source Control**, select **Launch Provider**. Describe launches your SCC provider application.

Refresh Status

To ensure that you can see the most current status of all elements under source code control, use the **Refresh Status** command on the Source Control menu.

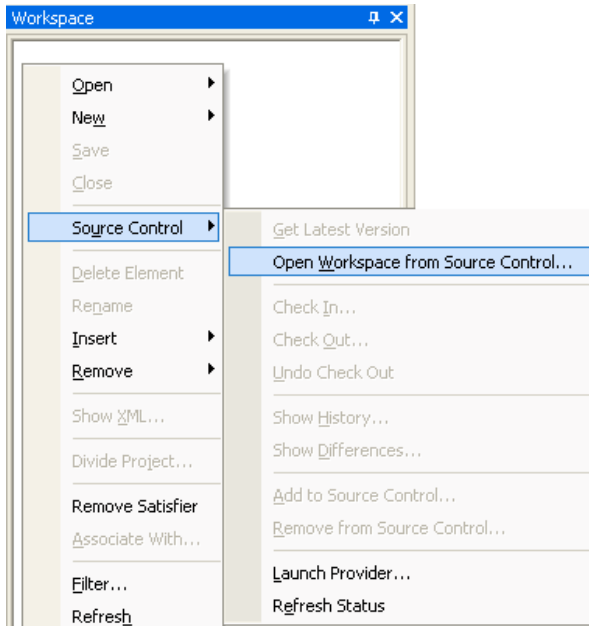
- Describe often refreshes the Workspace View pane. However, you may want to make sure that an element under source code control reflects its current status.
- Refreshing the status of elements under source code control is particularly useful when you work in a collaborative environment. This ensures that none of your team members already have an element checked out.

Open a Workspace from CM Tool

You can open an existing Workspace from the source code control by right clicking on the context menu in the Workspace pane **Source Control >Open Workspace from Source Control**.

- In the Workspace Pane right-click, and under **Source Control**, click **Open Workspace from Source Control**. This prompts you to log into your CM Tool and select the workspace you want.

- Once you have opened the workspace with corresponding projects, you can also create new projects, insert existing projects, reverse engineer new projects into a workspace, and create a new project via the ER/Studio integration.



Tips and Tricks while working with Configuration Management using Describe

When working in a collaborative manner, it is best to break projects into packages. This gives you the flexibility to version at the package level. You are able to create your own diagrams with their own modeling elements scoped to their inherent package, thus eliminating merging conflicts.

Best Practice Scenarios

Versioning at the Package Level Example

Project Lead A and Developers B, C & D are collaborating on one single project XYZ. Project XYZ exists inside of workspace XYZ.

- Project Lead A creates and versions an empty workspace XYZ, Project XYZ, and a package for each perspective developer.
- Once the workspace and project have been versioned, Developers B, C & D can now “Get Latest” on the workspace XYZ and project XYZ (which contains the corresponding package structure).
- Each user can now work within their assigned or owned package under the project and add elements to source code control.
- At any time, you can right click on the project XYZ and select “Get Latest Version”. This queries the CM Tool for new changes and

updates to the entire project XYZ (this moves the latest files to your local working directory).

NOTE: Until the SP1 release, you need to launch your CM provider and manually get a latest update to your working directory.

- Once this is done, go back to Describe, right-click on the project XYZ and select “Refresh”. This brings in newly created diagrams from the other users.
- Once this process is established, you can continue to work in a collaborative manner using your CM tool for the repository.

NOTE: If you are working in class diagrams and want to generate source code as you model, open the preconfigured “Configuration Management” dialog box from **Edit > Preferences** to make sure Describe is versioning the corresponding source code. For example, class “Customer” has an associated source code artifact “Customer.java” for example.

Sharing Elements across Packages Example

Developers/Modelers A, B, C, & D need to be able to share modeling elements across packages. The best practice for this environment is explained below.

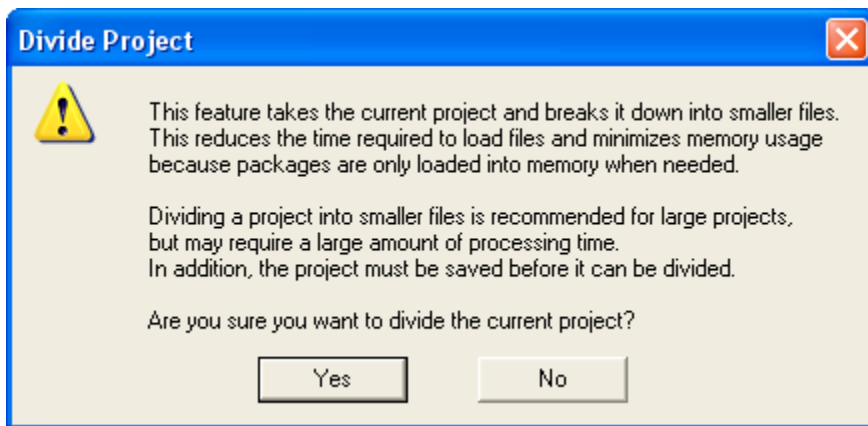
- Project Lead A creates and versions workspace XYZ, project XYZ, and package structure.
- Developer B, C, & D are able to “Get Latest Version” to their local working directories. Since all users are creating diagrams and using modeling elements from all existing packages, they need to version all packages, diagrams, and modeling elements.
- This enables you to turn on file locking if your CM tool provides that functionality, so that only one copy of each file can be checked out at one time. This ensures that you are not forced to merge models.
- If the CM tool does not provide this, you are able to check out items from CM tool and when you check them in, if there are other versions that exist, will merge them for an up-to-date latest version.
- When you version all items separately in a project you need to be aware of scenarios that may require you to check out the entire package and/or project. A potential scenario would be if a user creates a new data type (the data type is created at the project level) and is required to check out a package and/or project.
- You need to check in the project and package so that it is now available in the CM Tool repository.

- You will run into merge conflicts if you check out and work on the same diagram simultaneously. It is suggested that each user have their own working diagram that they check out and add elements to. A master diagram that contains everything is then managed by the Project Lead A.

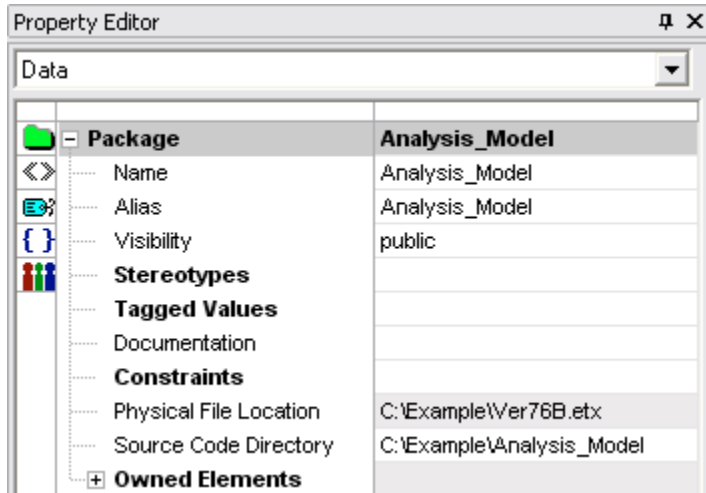
Dividing Large Projects

Describe lets you divide projects. When you divide a project, Describe takes the current project and breaks it down into smaller files. This reduces the time required to load the project and minimizes memory usage, because packages are only loaded into memory when needed.

Dividing a project into smaller files is recommended for large projects, but might require a large amount of processing time. In addition, you must save a project before dividing.



- In the **Workspace** pane, right-click the target project, and then click **Divide Project**. Describe opens a Divide Project message box.
- In the message box, click **Yes** to divide the project.
- Describe closes the message box, and divides the project into smaller files.
- The “Divide Project” mechanism is very similar in nature to versioning packages. Describe creates a *.etx file for each existing package in the Project. You can see the physical file location in the Property Editor.

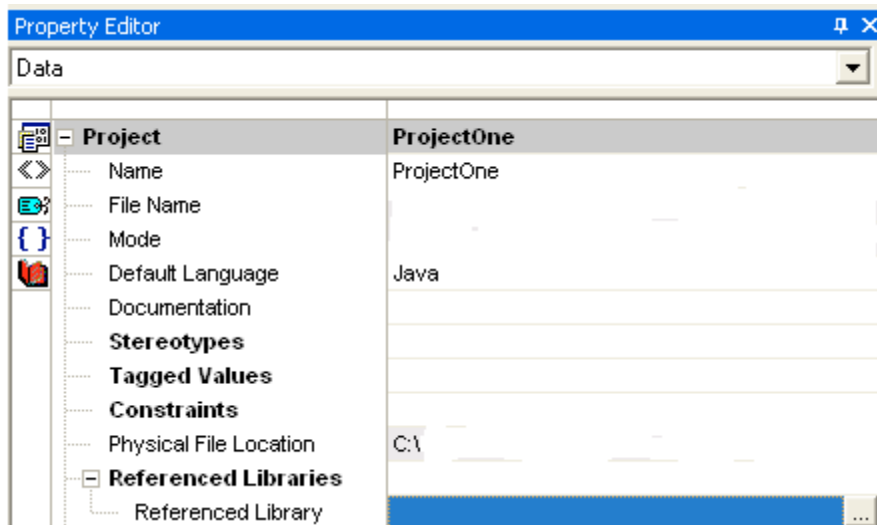


- This allows Describe to only load in memory the packages and scoped Meta data.
- Once a project is divided, you can version each package and scoped Meta data in the project for collaboration.

References Libraries

- What is a Reference Library?
 - A Reference Library is a Describe Project (*.etd) with the ability to use Default Libraries or User Defined Libraries.
 - For instance, if you choose to use a Reference Library you can reference these libraries with each project from the Property Editor using the “Reference Libraries” field and version the Project XYZ.
 - A Reference Library associated with a versioned project enables you to drag/drop these reference classes onto existing class diagrams.
 - You will also be able to double click on a return type (a “clickable” underscore appears under the return type) in a class and reference all the Library Classes and data types.
 - Using Reference Libraries eliminates merging conflicts when using common classes and data types throughout the Project.

NOTE: Select the project in the tree and the click on the Reference Libraries button as seen below. Once you have selected the Project in the Workspace pane, look at the Properties Editor and expand the Referenced Libraries node.



Extended Versionable Elements for Power Users

An extended Versionable element is an element that Describe does not include in the Versionable elements section in the Preference Editor. An example would be the ability to version a data type.

The following example enables you to add additional elements available for versioning control.

Configuration files that need to be modified.

- PreferenceProperties.etcd
- PreferenceProperties.etc

There is an additional Configuration file that Describe references for Naming Conventions. This Configuration file is called EssentialConfig.etc. You can open this file to view available elements for customizations.

NOTE: The files are located %installdir%\config.
For example, (C:\Program Files\Embarcadero\Describe\config).

The PreferenceProperties.etcd file contains the information provided in the Preference Dialog. The following is a snippet tag for Versionable elements.

```
<aElement name="VersionableElements">
  <aElement name="Actor" value="PSK_YES"/>
  <aElement name="AssociationClass" value="PSK_YES"/>
  <aElement name="Class" value="PSK_YES"/>
  <aElement name="Interface" value="PSK_YES"/>
  <aElement name="Package" value="PSK_YES"/>
  <aElement name="Project" value="PSK_YES"/>
  <aElement name="UseCase" value="PSK_YES"/>
  <aElement name="Activity" value="PSK_YES"/>
</aElement>
```

```
<aElement name="Interaction" value="PSK_YES"/>  
  <aElement name="StateMachine" value="PSK_YES"/>  
</aElement>
```

The PreferenceProperties.etc file contains the unique name and value for each of the Versionable elements. The following is a snippet tag for Versionable elements' name and value.

The PreferenceProperties.etc allows you to modify valuables and add custom text that is displayed in the Preference Dialog.

```
<aDefinition name="VersionableElements" displayName="PSK_VERSIONABLE_ELEMENTS"
helpText="PSK_VERSIONABLE_ELEMENT_HLP" image="\images\VersionableElements.ico">
<aDefinition name="Actor" displayName="PSK_VERSION_ACTOR_ELEMENTS"
controlType="list" values="PSK_YES|PSK_NO" defaultValue="PSK_YES"
helpText="PSK_VERSION_CONTROL_ACTORS_HLP"/>
<aDefinition name="AssociationClass"
displayName="PSK_VERSION_ASSOCIATIONCLASS_ELEMENTS" controlType="list"
values="PSK_YES|PSK_NO" defaultValue="PSK_YES"
helpText="PSK_VERSION_CONTROL_ASSOCIATION_CLASSES_HLP"/>
<aDefinition name="Class" displayName="PSK_VERSION_CLASS_ELEMENTS"
controlType="list" values="PSK_YES|PSK_NO" defaultValue="PSK_YES"
helpText="PSK_VERSION_CONTROL_CLASSES_HLP"/>
<aDefinition name="Interface" displayName="PSK_VERSION_INTERFACE_ELEMENTS"
controlType="list" values="PSK_YES|PSK_NO" defaultValue="PSK_YES"
helpText="PSK_VERSION_CONTROL_INTERFACES_HLP"/>
<aDefinition name="Package" displayName="PSK_VERSION_PACKAGE_ELEMENTS"
controlType="list" values="PSK_YES|PSK_NO" defaultValue="PSK_YES"
helpText="PSK_VERSION_CONTROL_PACKAGES_HLP"/>
<aDefinition name="Project" displayName="PSK_VERSION_PROJECT_ELEMENTS"
controlType="list" values="PSK_YES|PSK_NO" defaultValue="PSK_YES"
helpText="PSK_VERSION_CONTROL_PROJECTS_HLP"/>
<aDefinition name="UseCase" displayName="PSK_VERSION_USECASE_ELEMENTS"
controlType="list" values="PSK_YES|PSK_NO" defaultValue="PSK_YES"
helpText="PSK_VERSION_CONTROL_USE_CASES_HLP"/>
<aDefinition name="Activity" displayName="PSK_VERSION_ACTIVITY_ELEMENTS"
controlType="list" values="PSK_YES|PSK_NO" defaultValue="PSK_YES"
helpText="PSK_VERSION_CONTROL_ACTIVITY_HLP"/>
<aDefinition name="Interaction" displayName="PSK_VERSION_INTERACTION_ELEMENTS"
controlType="list" values="PSK_YES|PSK_NO" defaultValue="PSK_YES"
helpText="PSK_VERSION_CONTROL_INTERACTION_HLP"/>
<aDefinition name="StateMachine"
displayName="PSK_VERSION_STATEMACHINE_ELEMENTS" controlType="list"
values="PSK_YES|PSK_NO" defaultValue="PSK_YES"
helpText="PSK_VERSION_CONTROL_STATEMACHINE_HLP"/>
</aDefinition>
```

Extended Versionable Element

This example demonstrates how you can modify the XML configuration files to version non-default Versionable elements listed in the Preference dialog. This example adds a “DataType” in the Preference dialog as well availability to version the element via the Workspace View Pane.

Steps

- Open the EssentialsConfig.etc file in a text editor.
- Navigate to the “Data Type” to see the unique naming convention.
 - In this case the unique naming convention is
- Open the PreferenceProperties.etc file in a text editor.
- Insert the “Data Type” at the end of the last Versionable elements tag as seen below.

```
<HiveElement name="DataType" id="Constructs.DataType"/>
```

```
<aElement name="DataType" value="PSK_YES"/>
```

The PreferenceProperties.etc file contains the information provided in the Preference Dialog. The following is a snippet tag for Versionable elements which includes “DataType”.

```
<aElement name="VersionableElements">
  <aElement name="Actor" value="PSK_YES"/>
  <aElement name="AssociationClass" value="PSK_YES"/>
  <aElement name="Class" value="PSK_YES"/>
  <aElement name="Interface" value="PSK_YES"/>
  <aElement name="Package" value="PSK_YES"/>
  <aElement name="Project" value="PSK_YES"/>
  <aElement name="UseCase" value="PSK_YES"/>
  <aElement name="Activity" value="PSK_YES"/>
  <aElement name="Interaction" value="PSK_YES"/>
  <aElement name="StateMachine" value="PSK_YES"/>
  <aElement name="DataType" value="PSK_YES"/>
</aElement>
```

- Open the PreferenceProperties.etc file in a text editor.
- Insert the “Data Type” at the end of the last Versionable elements tag as seen below.

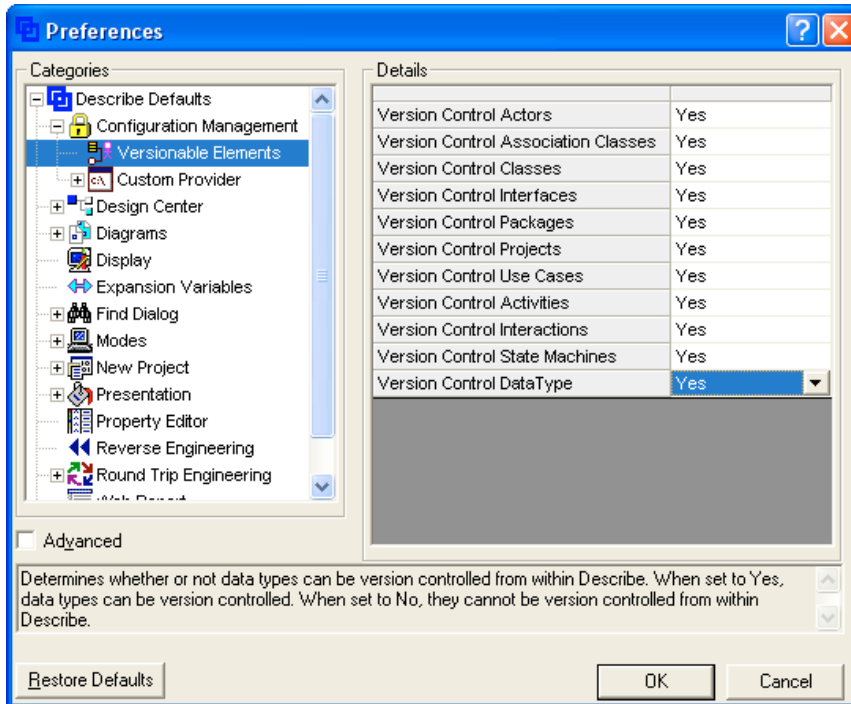
```
<aDefinition name="DataType" displayName="PSK_VERSION_ DataType "
controlType="list" values="PSK_YES|PSK_NO" defaultValue="PSK_YES"
helpText="PSK_VERSION_CONTROL_DataType_HLP"/>
```

The PreferenceProperties.etc file contains the unique name and value for each of the Versionable elements. The following is a snippet tag for Versionable elements' name and value.

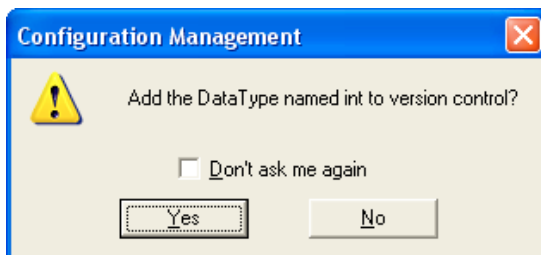
```
<aDefinition name="VersionableElements" displayName="PSK_VERSIONABLE_ELEMENTS"
helpText="PSK_VERSIONABLE_ELEMENT_HLP" image="\images\VersionableElements.ico">
<aDefinition name="Actor" displayName="PSK_VERSION_ACTOR_ELEMENTS"
controlType="list" values="PSK_YES|PSK_NO" defaultValue="PSK_YES"
helpText="PSK_VERSION_CONTROL_ACTORS_HLP"/>
<aDefinition name="AssociationClass"
displayName="PSK_VERSION_ASSOCIATIONCLASS_ELEMENTS" controlType="list"
values="PSK_YES|PSK_NO" defaultValue="PSK_YES"
helpText="PSK_VERSION_CONTROL_ASSOCIATION_CLASSES_HLP"/>
<aDefinition name="Class" displayName="PSK_VERSION_CLASS_ELEMENTS"
controlType="list" values="PSK_YES|PSK_NO" defaultValue="PSK_YES"
helpText="PSK_VERSION_CONTROL_CLASSES_HLP"/>
<aDefinition name="Interface" displayName="PSK_VERSION_INTERFACE_ELEMENTS"
controlType="list" values="PSK_YES|PSK_NO" defaultValue="PSK_YES"
helpText="PSK_VERSION_CONTROL_INTERFACES_HLP"/>
<aDefinition name="Package" displayName="PSK_VERSION_PACKAGE_ELEMENTS"
controlType="list" values="PSK_YES|PSK_NO" defaultValue="PSK_YES"
helpText="PSK_VERSION_CONTROL_PACKAGES_HLP"/>
<aDefinition name="Project" displayName="PSK_VERSION_PROJECT_ELEMENTS"
controlType="list" values="PSK_YES|PSK_NO" defaultValue="PSK_YES"
helpText="PSK_VERSION_CONTROL_PROJECTS_HLP"/>
<aDefinition name="UseCase" displayName="PSK_VERSION_USECASE_ELEMENTS"
controlType="list" values="PSK_YES|PSK_NO" defaultValue="PSK_YES"
helpText="PSK_VERSION_CONTROL_USE_CASES_HLP"/>
<aDefinition name="Activity" displayName="PSK_VERSION_ACTIVITY_ELEMENTS"
controlType="list" values="PSK_YES|PSK_NO" defaultValue="PSK_YES"
helpText="PSK_VERSION_CONTROL_ACTIVITY_HLP"/>
<aDefinition name="Interaction" displayName="PSK_VERSION_INTERACTION_ELEMENTS"
controlType="list" values="PSK_YES|PSK_NO" defaultValue="PSK_YES"
helpText="PSK_VERSION_CONTROL_INTERACTION_HLP"/>
<aDefinition name="StateMachine"
displayName="PSK_VERSION_STATEMACHINE_ELEMENTS" controlType="list"
values="PSK_YES|PSK_NO" defaultValue="PSK_YES"
helpText="PSK_VERSION_CONTROL_STATEMACHINE_HLP"/>
<aDefinition name="DataType" displayName="Version Control Data Type" controlType="list"
values="PSK_YES|PSK_NO" defaultValue="PSK_YES" helpText="Enter comments here for
documentation"/>
</aDefinition>
```

- Adding documentation to Preference Dialog using XML tagged values
 - You can modify the tag values to display custom information in the Preference Dialog.
 - For example, the helpText value for versioning Data types may read “Determines whether or not data types can be version controlled from within Describe. When set to Yes, data types can be version controlled. When set to No, they cannot be version controlled from within Describe.”

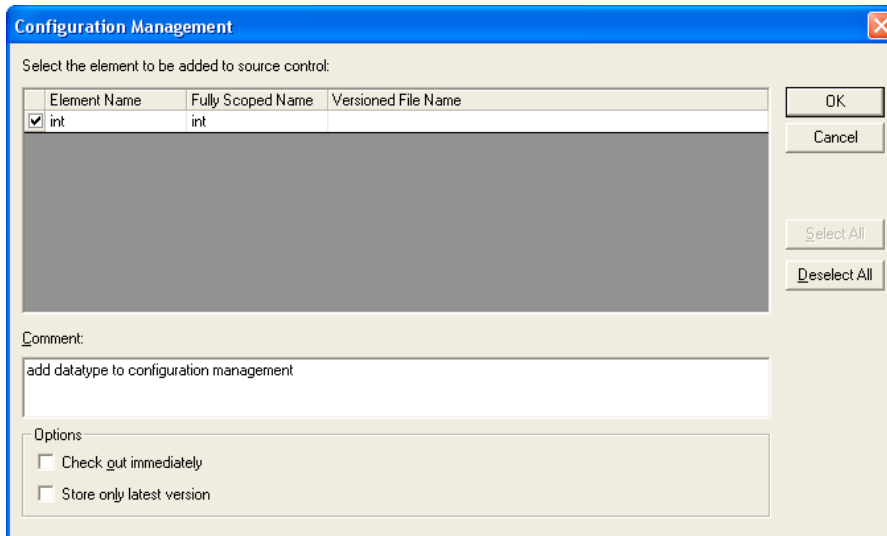
The following Preferences dialog displays the changes made to allow you to version Data types. Notice the changes made to the newly created “Version Control Data Type” field and the associated help.



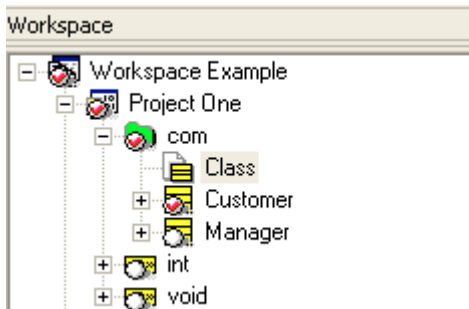
When new data types are created, Describe prompts you to add the element to Version Control.



The following Configuration Management Dialog displays the common functionality to version the data type.



The data type node now represent that the element is versioned by displaying the versioning icons via the Workspace View pane.



Conclusion

Describe enables collaboration via its integration with Configuration Management Applications. This white paper has outlined Best Practices and examples of how to be successful using Describe in a team environment. You also have been given an example of how to extend versionable elements for configuration management. In conclusion, you will be successful with team collaboration once you understand the architecture and integration with Configuration Management applications.