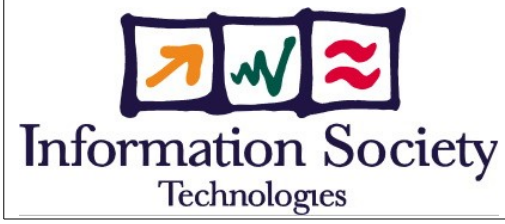
Digital Business Ecosystem

Contract n° 507953

Workpackage 26: DBE Studio/DBE Portal

Deliverable D26.3: DBE Studio Integration



Project funded by the European Community under the “Information Society Technology” Programme

Contract Number: 507953

Project Acronym: DBE

Title: Digital Business Ecosystem

Deliverable N°: 26.3

Due dates: 12/2005

Delivery Date: 04/2006

Short Description:

The DBE Studio is an Integrated Development Environment (IDE) for the Digital Business Ecosystem (DBE). It includes Eclipse plugins that allow business services to be analysed, and corresponding software services to be defined, developed and deployed. The DBE Studio Integration task requires the ongoing managing of component integration and release builds, along with core plugin development.

This document outlines these ongoing tasks, and complements the actual software and websites which also form part of this deliverable. A developer's integration guide is reproduced in an appendix.

Author: Intel Ireland Ltd.

Partners contributed: Intel Ireland Ltd.

Made available to: Public

Versioning		
Version	**Date**	**Author, Organisation**
0.1	04/02/2006	David McKitterick, Intel Ireland
0.2	22/02/2006	David McKitterick, Intel Ireland
0.3	06/03/2006	David McKitterick, Intel Ireland
0.4	15/03/2006	David McKitterick, Intel Ireland

Quality check:

1st Internal Reviewer : Pierfranco Ferronato, Soluta.Net

2nd Internal Reviewer: Fotis G. Kazasis, TUC

TABLE OF CONTENTS

1. Introduction

The DBE Studio is an Integrated Development Environment (IDE) for the Digital Business Ecosystem (DBE). The DBE is a semantically-aware, service oriented architecture for Small and Medium sized Enterprises (SMEs). The architecture of the DBE is divided into three main logical components: Service Factory, Execution Environment (ExE) and Evolutionary Environment (EvE). The Service Factory contains a set of tools and core services which enable the creation and modeling of business and software services. The ExE is a runtime environment where services can be deployed and executed by users and other services. The EvE uses advanced algorithms to help optimise the results of DBE queries by intelligently migrating service representations through the ExE .

The DBE Studio is the implementation of the Service Factory. It is built on top of the Eclipse platform [6] where each tool is implemented as an Eclipse plugin. These tools allow business services to be analysed and modeled based on a business modeling language, and is where the corresponding software services can be defined, developed and deployed using a set of editors and wizards provided by the aggregation of plugins. The Eclipse platform provides an extensible pluggable architecture which ideally suits the requirements of an evolving development environment. The DBE Studio requires additional Eclipse frameworks (e.g. graphical and modeling frameworks) to the standard Eclipse distribution and it is intended that users will make use of the standard Java development tools also available with Eclipse.

The target users of the DBE Studio are advanced end users and software developers. These may vary from business analysts, who need to model their business and software services using the business modeling tools, to SME software developers, who need to expose existing legacy systems as a software service using the platform independent development and deployment tools.

The development of DBE Studio is a collaborative effort by many DBE partners which therefore requires effective coordination of this integrated process along with the additional interconnections with other DBE environments such as the DBE Servent. The DBE Studio Integration task requires the ongoing managing of component integration and release builds along with core plugin development. This task was created to provide a means to efficiently establish and coordinate this open source project within the SourceForge domain.

This document complements the software deliverable for the DBE Studio Integration task. Although the work under this task is ongoing and recurring in nature, some significant

achievements have been accomplished. These include the initial creation of the DBE Studio SourceForge project, extensive coordination of component integration and release builds, development of some core plugins and the ongoing delivery of help and support to developers and users of the DBE Studio.

2. DBE Studio on SourceForge.net

The DBE Studio project [1] was established within the SourceForge group [2] in July 2005. SourceForge is an open source software development web site, and with over 100,000 projects and over 1 million registered users, is the largest of its kind. SourceForge provides online resources for managing open source projects, including bug trackers, mailing lists, forums, and a file release system. As the largest open source code repository, SourceForge also hosts the largest amount of open source development products and applications available on the Internet.

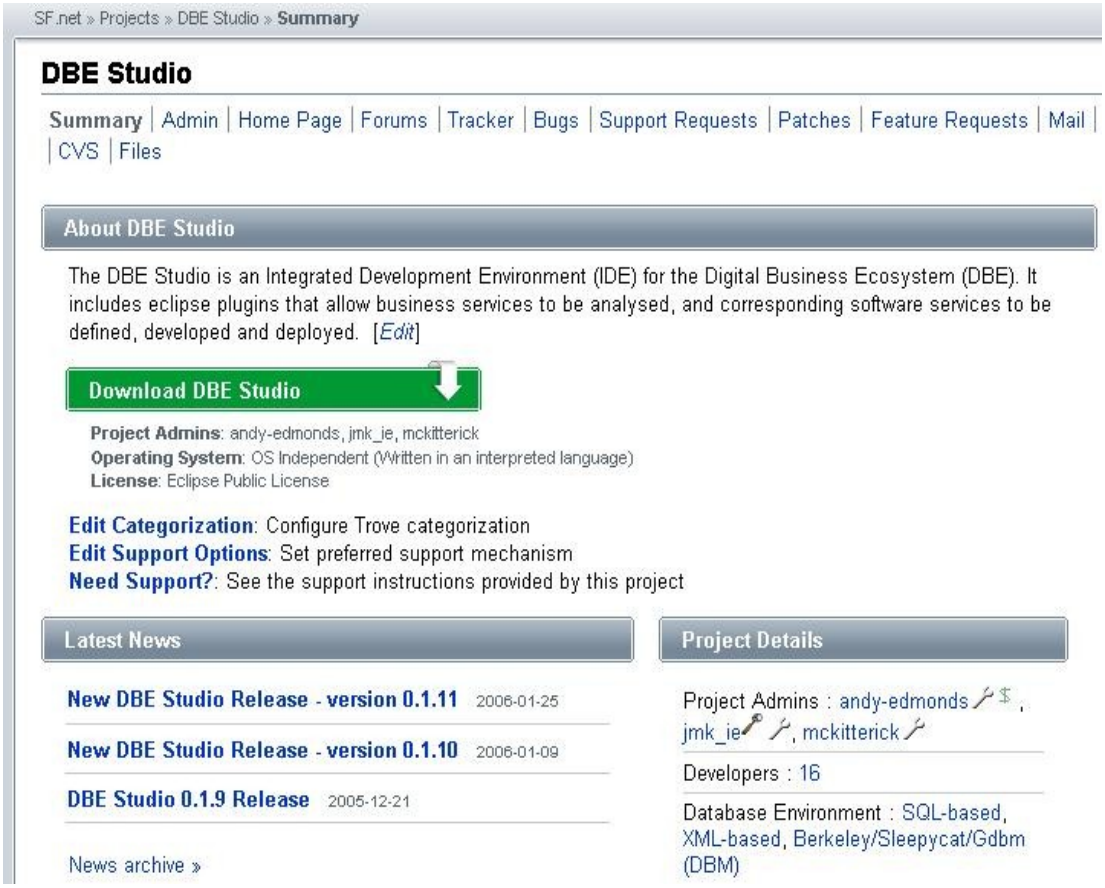


Figure 1: http://sourceforge.net/projects/dbestudio/

The DBE Studio SourceForge project was setup as part of the DBE Studio Integration task (see Figure 1 for the DBE Studio project on SourceForge). One of the initial tasks when creating this project was to decide upon an open source license which best suited the long term goals of the DBE project. After extensive consultation with both project partners and open-source legal experts, the Eclipse Public License (version 1.0) [3] was selected as the open source license for all DBE Studio source code and software releases. This license reflects the close dependency with the Eclipse platform and associated frameworks, and also does not restrict the further development of DBE Studio based tools by external parties.

Using the project facilities provided by SourceForge, a project home page [4] was created (see Figure 2). This informs visitors and active users of the project's installation procedures and development status. The web site also provides access to the support features of this project, such as mailing lists, forums and bug trackers. Given that numerous components are being developed by different people within the DBE Studio project, users can target their bug, feature or support requests directly at the developer(s) responsible for the relevant plugin. There is also a link on the home page to our integrated build system web site which will be explained in more detailed in chapter 4, 'Build and Release Management'. SourceForge provides a CVS repository for each project - a necessary facility for integrated open source projects. All DBE Studio source code is available for browsing and downloading by anyone, but CVS commit rights are restricted to allow only active project developers to modify and add files. CVS administration for this project will be explained further in chapter 3, 'Component Integration'.

Figure 2: http://dbestudio.sourceforge.net

3. Component Integration

The DBE Studio currently comprises of 22 separate Eclipse plugins being developed by 16 developers from 7 different partners within the DBE consortium. This large mix of components and developers requires highly coordinated integration procedures. As part of the DBE Studio Integration task, an integration guidelines document for DBE Studio project developers was released. This document outlines the recommended procedures for creating a plugin project within the project's CVS repository, integrating a plugin project into the self-contained Maven build system, using CVS with Eclipse and using common versioning techniques. The latest version of this document can be found in appendix A, 'DBE Studio-Developer's Integration Guide'.

The component structure of the DBE Studio has been divided into three sections: core, editors and tools. This structure is reflected in the CVS repository and the naming convention used for plugin projects, e.g. '*org.dbe.studio.editors.bml*'. A summary of components within the DBE Studio are listed in Table 1. Each component group is described as follows:

- A core component is generally a plugin which is central to the DBE Studio distribution and is used by some other DBE Studio plugins. Core plugins include *DBE Studio Preferences* and *DBE Studio Perspectives*.
- An editor component is a plugin which provides an Eclipse editor view for creating and editing files and models, such as the *BML* and *SDL* editors.
- A tool component is any plugin which does not provide an editor view or is not considered a core plugin within the DBE Studio. For example the *Service Exporter* tool.

Once the above structure was established, help was provided to most of the relevant partners with the task of creating their plugin projects. This assistance included such things as setting up the correct structure of their projects in conformance with the integrated build system, help with CVS operations and usage, advice on plugin development and UI techniques, and help with integration to other plugins (Eclipse and DBE Studio). A set of UI icons was also created and provided for the majority of plugin wizards and editors. This was accompanied with advice on best known approaches for UI tool design.

During the development of plugins for the DBE Studio, ongoing analysis was performed on the functionality provided by all plugins. The overall testing process involved testing individual plugins, and if problems were observed, a communication process was initiated

with the partner or particular developer responsible for this plugin. This communication initially was done by email but following the early stages of the SourceForge project, most bug related issues were posted on the online bug tracker provided by SourceForge.net.

Component Groups	Component Name	Plugin ID	Contributing Partner(s)
Core	Feature	*org.dbe.studio.core.feature*	Intel Ireland
	Perspectives	*org.dbe.studio.core.perspectives*	Intel Ireland
	Preferences	*org.dbe.studio.core.preferences*	Intel Ireland
	Studio Examples	*org.dbe.studio.core.examples*	Intel Ireland
	Studio Help	*org.dbe.studio.core.help*	Intel Ireland, Others
	UI	*org.dbe.studio.core.ui*	Intel Ireland
Editors	BML	*org.dbe.studio.editors.bml*	TUC
	BML Data	*org.dbe.studio.editors.bmldata*	UCE
	BPEL	*org.dbe.studio.editors.bpel*	TCD
	ODM	*org.dbe.studio.editors.odm*	TUC
	SDL	*org.dbe.studio.editors.sdl*	Soluta
Tools	KB Toolkit	*org.dbe.studio.tools.kbtoolkit*	TUC
	Metering Wizard	*org.dbe.studio.tools.accounting.metering*	WIT
	Ontology Viewer	*org.dbe.studio.tools.ontologyviewer*	TUC
	QF-SDT	*org.dbe.studio.tools.qfsdt*	TUC
	Recommender	*org.dbe.studio.tools.recommender*	TUC
	SDL2Java Compiler	*org.dbe.studio.tools.sdl2java*	Soluta
	SDL Storage	*org.dbe.studio.tools.sdl.storage*	Soluta
	Service Exporter	*org.dbe.studio.tools.exporter*	Intel Ireland
	SM Creator	*org.dbe.studio.tools.smcreator*	Soluta
	SSL2SDL Compiler	*org.dbe.studio.tools.ssl2sdl*	Soluta
	Test Case Generator	*org.dbe.studio.tools.testcasegenerator*	UniS

Table 1: Component List for DBE Studio

To further improve the integration between DBE Studio plugins, some core plugins, as previously mentioned, were developed and integrated into this increasing set of tools. These plugins provided core functionality such as aggregating user preferences in a common location for all DBE Studio plugins, providing a general user help guide where all plugin user guides could be located, and creating a consistent set of task-oriented perspectives where related tools were visually connected. These core plugins will be explained in more detail in chapter 5, ‘Core Plugin Development’.

4. Build and Release Management

One of the most important tasks of DBE Studio Integration was to create a unified mechanism to build regular releases of the DBE Studio. This task can be separated into two parts: build management and release management. Build management involves integrating and administrating an effective build system for all component projects in the CVS repository, enabling a unified build and distribution of the software product. Release management involves creating and administrating a mechanism which supports regular user-friendly install and update procedures for the released software.

Maven as an Integrated Build System

As already used in the previous phase of project, it was decided to continue using Maven [5] as the open source build system for the DBE Studio. In addition to a build system, Maven provides software project management and comprehension tools. Maven helps project integrators to build project components as part of unified builds, create project reporting information and present this information using an automatically generated web site. Extra Maven components were modified and added to the standard build system, including functionality to improve the building of Eclipse plugins automatically from Maven projects.

Figure 3: http://dbestudio.sourceforge.net/maven-site/main/docs/

The integrated build system provided by Maven was uploaded to the CVS repository so that all developers could have a standalone build system for the DBE Studio on their local machines. Each DBE Studio plugin project was created as a Maven project within the CVS structure, as described in chapter 3. Therefore by building each Maven project it was possible to automatically build an Eclipse plugin which was ready for distribution. Each plugin project requires a project.xml file which declares the project's id, version number and dependencies, along with other general information. Various Maven commands are available to compile the Java source code, run unit and component tests, build an Eclipse plugin and also generate a project web site. A build script was created so that a developer can simply run this script to build all components and copy the generated Eclipse plugins to a common directory location. The generated web site, as shown in Figure 3, provides information such as plugin project dependency lists, code convention correctness reports, test results, versioning and change logs.

A Maven repository was setup on the DBE Studio SourceForge project web site to contain all versioned jars of the DBE Studio plugin classes. This repository also contains third-party open source jar dependencies which are used by the DBE Studio Maven projects.

DBE Studio Install and Upgrade Mechanism

SourceForge provides a file release system where files can be uploaded and mirrored across the SourceForge network. These file releases are available for download by users from the project's SourceForge site. As the DBE Studio is built on top of the Eclipse platform and each component within the DBE Studio is distributed as an Eclipse plugin, it seemed obvious to follow the Eclipse install and update mechanism with which existing Eclipse users would be already familiar. This required the creation of an update site which specified the DBE Studio feature version and the required plugins for this feature. For both first time users and existing users of the DBE Studio, a simple procedure enables them to manually (or automatically) install or update the latest version of the DBE Studio from within their Eclipse installation. Ensuring that the required feature dependencies were installed, a first time user can search for a DBE Studio feature using the Eclipse 'Software Updates, Find and Install' mechanism. After entering the update site url for the DBE Studio, i.e. http://dbestudio.sourceforge.net/install, they can select the latest feature and install it within their existing Eclipse installation.

Another available procedure involves downloading an archive of the DBE Studio distribution

from the SourceForge file release system and then using the Eclipse software update mechanism to install this local archive. See Figure 4 for the Eclipse software update wizard displaying the DBE Studio feature. For existing users of the DBE Studio, it is possible to either manually search for new feature releases or to set up automatic updates which will automatically search for new feature releases and install the update if desired by the user.

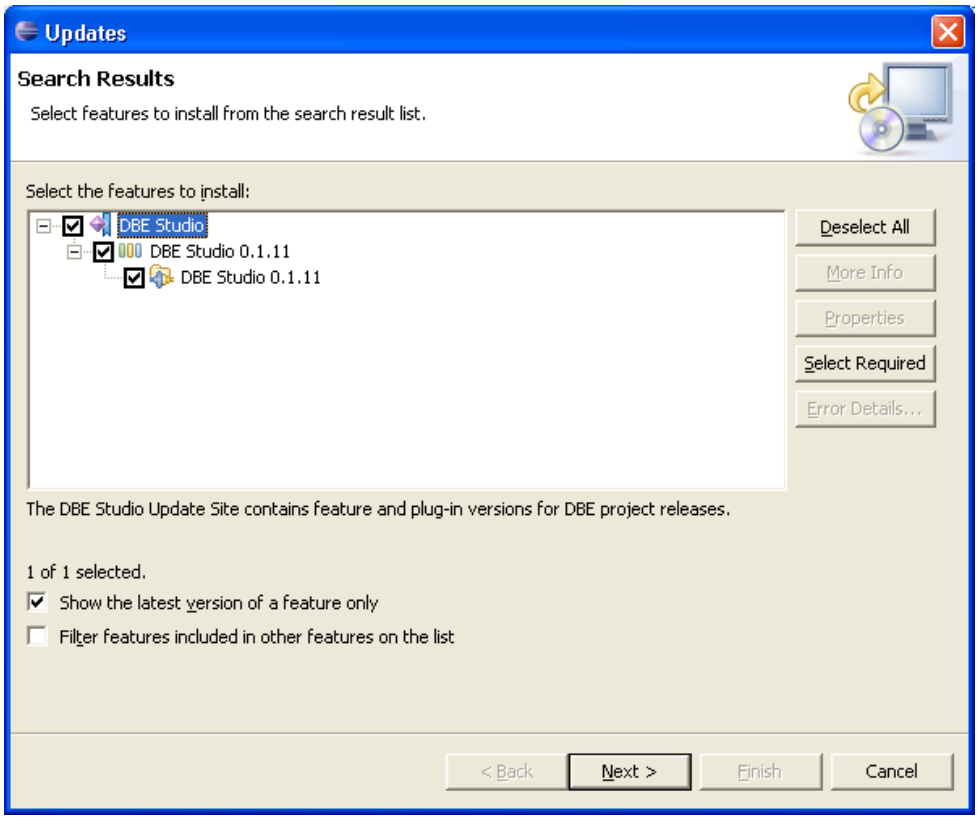


Figure 4: DBE Studio Install/Update Mechanism

5. Core Plugin Development

As part of the DBE Studio Integration task, several core plugins have been developed and contributed to the DBE Studio distribution. These plugins provide functionality for feature and UI support, user preferences, user help guides and examples, and a tool to deploy DBE services to the DBE Execution Environment. The following sections describe these plugins in more detail.

DBE Studio Feature

Most products based on the Eclipse platform are distributed as features. A feature describes the distribution of the product by listing the plugins included with this feature and the required dependencies of the feature and its plugins. The feature enables efficient versioning of different distributions. See Figure 5 for the DBE Studio Feature within Eclipse.

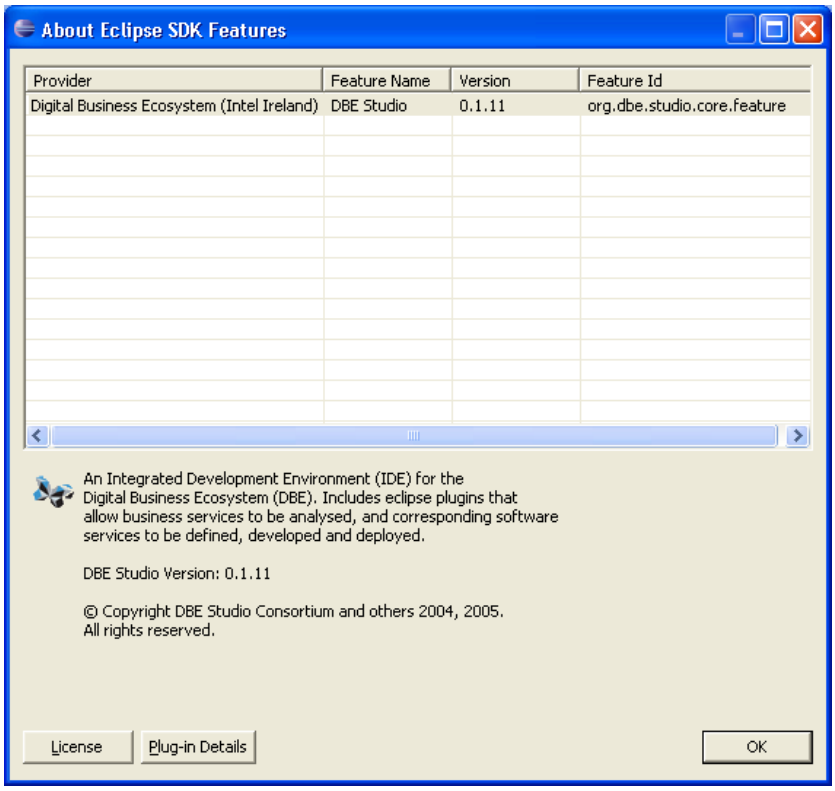


Figure 5: DBE Studio Feature within Eclipse

Examples

The example plugin, which is distributed separately to the DBE Studio distribution, provides DBE projects containing example DBE services. These projects contain the relevant source files and models for analysing, building and deploying the services.

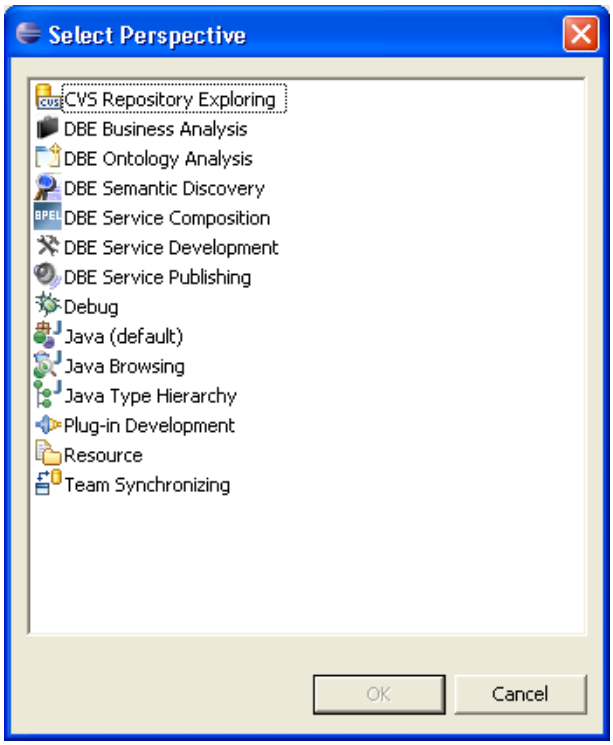


Figure 6: List of DBE Studio Perspectives within Eclipse

Perspectives

This plugin provides a set of perspectives for the DBE Studio. Examples of perspectives included are *DBE Service Publishing* and *DBE Service Development*. A perspective declares an arrangement of a set of related views and editors which are combined together to perform specific tasks. The DBE Studio perspective plugin also provides the concept of a DBE project. A user can create a new DBE project which provides a directory structure for all DBE files. This structure is also used by the *Service Exporter* tool for deploying services to a DBE Servent. Figure 6 shows a list of all the DBE Studio perspectives within Eclipse.

Preferences

The preferences plugin provides an entry in the Eclipse workspace preference menu for global DBE Studio preferences. This plugin also provides an extension point for plugin developers to add tool related preferences to the DBE Studio preference list. Figure 7 shows the DBE Studio global preferences page and its subcategory pages for individual tool preferences.

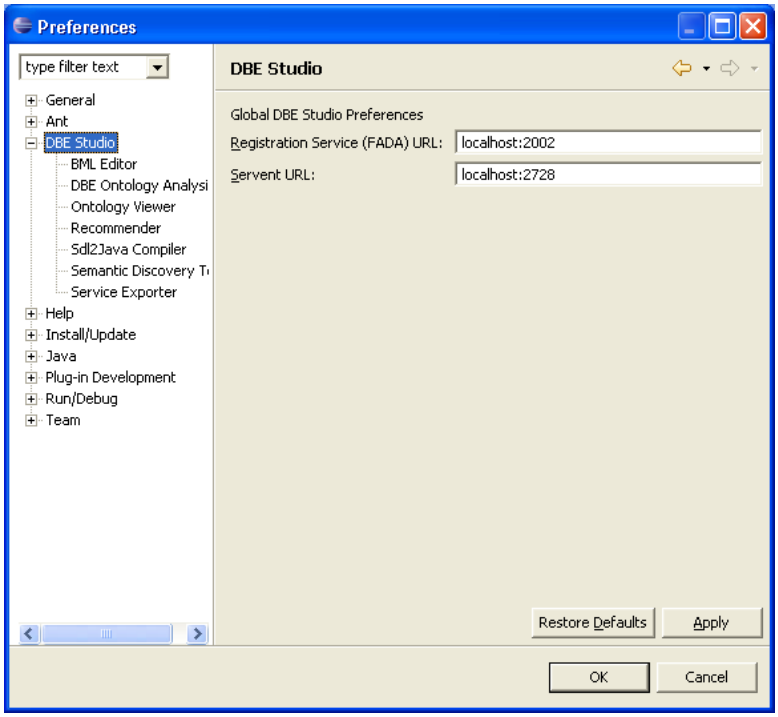


Figure 7: DBE Studio Preferences Dialog

Service Exporter

The Service Exporter tool is implemented as an export wizard and allows a selected DBE project to be exported to the DBE. This involves the service implementation (and optional UI) within the DBE project being added to a DAR (DBE Archive) file and then being deployed to a particular Servent. The Service Exporter also supports multiple UI types, composed DBE services, and adding Servent filters and properties to services. This plugin is also integrated with the Accounting Metering Wizard which enables users to specify metering filters for their deployed services. Figure 8 shows the DBE Service Exporter tool listed within Eclipse's export wizard options.

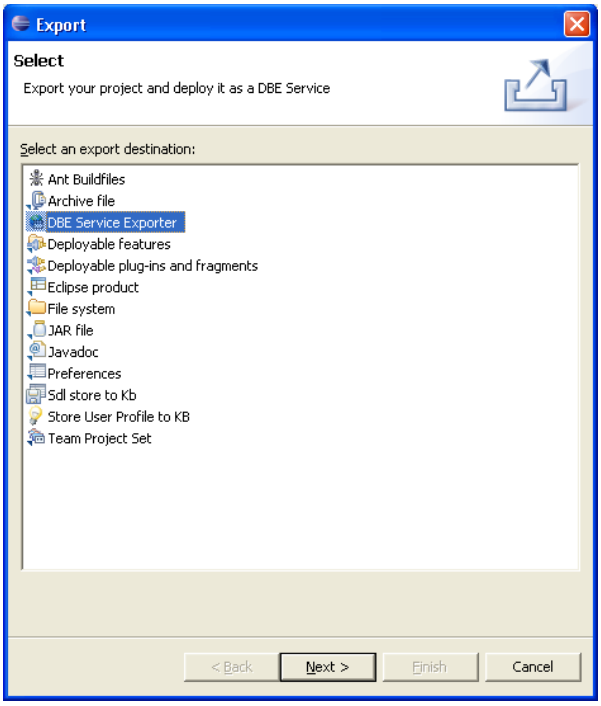


Figure 8: DBE Service Exporter

User Interface (UI)

This is a branding plugin for the DBE Studio feature which provides branding details related to the feature and its plugins, such as licensing and contributing partner information. It is intended that this plugin will be extended to provide additional UI support functionality to the DBE Studio.

Help

The help plugin contains a set of user guides for most tools within the DBE Studio which can easily be accessed via the main Eclipse help menu. These user guides were provided by the contributing partners within the DBE Studio SourceForge project. All documentation was written in DocBook [7], an XML based document type, which enabled easy integration and the generation of both html and pdf formated files. In addition, a Cheat Sheet is provided with this plugin. This Cheat Sheet is an interactive tutorial which opens a view on the right hand side of the screen outlining steps on how to get started with the DBE Studio tools and therefore creating and deploying a DBE Service. Figure 9 shows the DBE Studio User Guide.

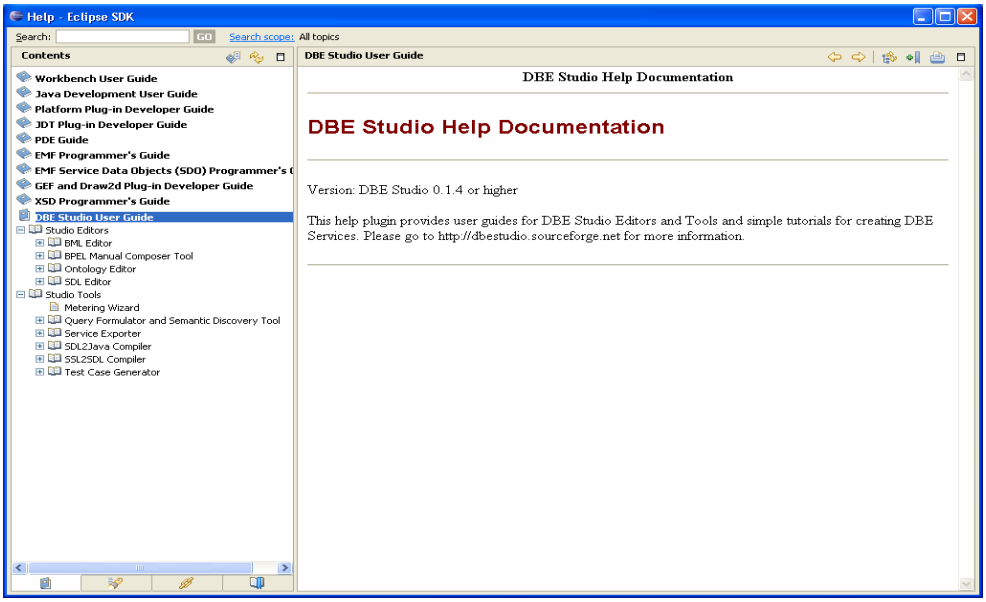


Figure 9: DBE Studio User Guide within Eclipse

6. Conclusion

As part of the DBE Studio Integration task a SourceForge project has been setup and administered, a project web site has been created, and a project structure and build system has been defined. An integration guideline document has been developed for contributors, and assistance has been provided throughout the lifecycle of DBE Studio plugin development to the contributing partners. The build, testing and release of the DBE Studio as well as the definition of an installation procedures has also been accomplished as part of this task. Finally, core DBE Studio plugins themselves have been developed, including Examples, Perspectives, Preferences and a Service Exporter plugin.

This task is ongoing in nature, and it is foreseen that numerous releases of DBE Studio will be supported during the remainder of the project. These releases will include newly integrated DBE Studio components (including a SBVR Editor), as well as support for any relevant upgrades of the DBE Studio dependencies, including Eclipse and Java. Several initiatives are also planned to enhance the build system and support for developers, including the deployment of a DBE Studio Wiki to which anyone in the DBE Studio community can contribute.

7. References

1. DBE Consortium, DBE Studio SourceForge Project, http://sourceforge.net/projects/dbestudio.
2. SourceForge.net, https://sourceforge.net/docs/about
3. OpenSource.Org, Eclipse Public License, http://www.opensource.org/licenses/eclipse-1.0.php
4. DBE Consortium, DBE Studio Home Page, http://dbestudio.sourceforge.net
5. Apache.org, Maven home page, http://maven.apache.org
6. Eclipse Open Source Community, http://www.eclipse.org/
7. DocBook Technical Committee Document Repository, http://www.oasis-open.org/docbook/

8. Appendix A: Developer's Integration Guide

DBE Studio – Developer's Integration Guide
Digital Business Ecosystem

Intel Ireland

Versioning

Date	Author	Version	Changes
12/07/2005	Intel Ireland Ltd.	0.1	Initial Draft
12/07/2005	Intel Ireland Ltd.	0.2	Added CVS Guidelines
16/08/2005	Intel Ireland Ltd.	0.3	Added process diagrams, First Release
12/09/2005	Intel Ireland Ltd.	0.4	Added eclipse CVS configuration. Added section on Maven
04/02/2006	Intel Ireland Ltd.	0.5	Additional upgrade

Introduction

This document is intended to outline some guidelines and rules for developers/partners contributing to the DBE Studio SourceForge.net project. The project can be accessed from:

- http://dbestudio.sourceforge.net/
- http://sourceforge.net/projects/dbestudio.

Details on becoming a SourceForge user or developer or any other specific SourceForge issues are out of the scope of this document. Please refer to the SourceForge website [1] for more information. This document will be divided into the following sections:

- Partners, Roles and Components.
- CVS Module Structure.
- CVS Procedures with Eclipse.
- Studio Plugin and Naming Conventions.
- Versioning.
- Plugin Distributions.

As a prerequisite to reading this document, all developers are advised to review a previous released document, the DBE Developers' Guide [2], written by Dominik Dahlem of Trinity College Dublin. We make the assumption that all developers are familiar with this document, especially with regards to code conventions, CVS, and the Maven [3] build system. We intend to continue using the Maven build system to manage the builds within this project and to conform to the conventions declared in the DBE developers' guide.

Partners, Roles and Components

In Figure 10, an organisational overview of the DBE Studio SourceForge project is presented (although this represents the initial status of the DBE Studio project. Additional partners and components have been included since this initial stage). At the lowest layer are the components (Eclipse [5] plugins), comprised in the DBE Studio. Moving up the diagram, the partners that are responsible for implementing these components are shown. At the next level, Intel is listed as being integrators of these components supplied by partners. The integrators have the responsibility of distributing a seamless and unified DBE Studio distribution to

SourceForge. This will involve administration of the whole CVS [4] root and Maven builds, development of a Studio installation and update mechanism, and any other integration tasks needed. Please note, Intel will not be responsible for managing and administrating each individual component; this is the responsibility of the implementing partner.

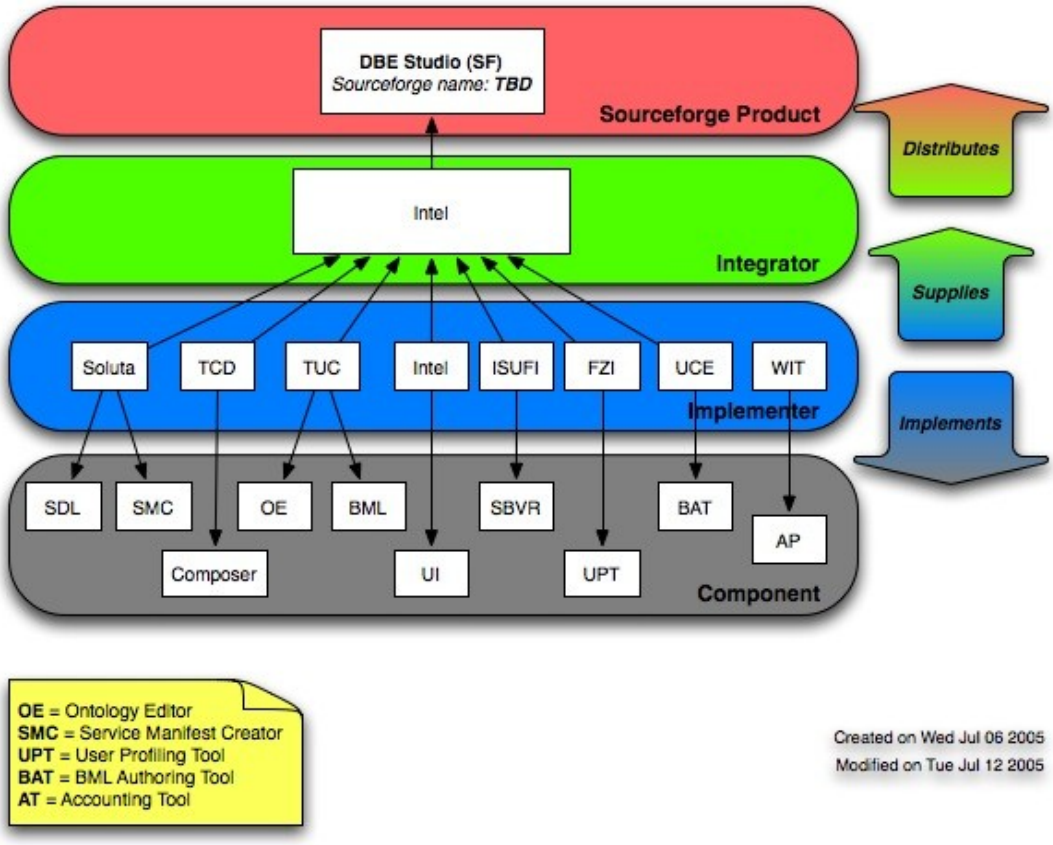


Figure 10: DBE Studio SourceForge (Initial) Project Overview

CVS Module Structure

From the organisational diagram in Figure 10, the following CVS root hierarchy (Figure 11) has been devised. The main CVS root structure has been divided into two main sections, Maven build system directories and DBE Studio source directories. More information about the Maven build system directories can be found in the DBE developers' guide and will no longer be discussed in this document. The DBE Studio source directories are split into three main modules: `studio-core`, `studio-editors` and `studio-tools`. **`studio-core`** is a module for any core components that are common to all studio plugins. This may include perspective plugins, a studio Eclipse feature, branding and DBE Studio help. **`studio-editors`** is a module for any component that provides an editor view within the Studio. Each editor component will only have one plugin. These conventions for plugins will be discussed later in this document. **`studio-tools`** is a module for any plugin which

provides a tool and does not provide an editor view within the Studio.

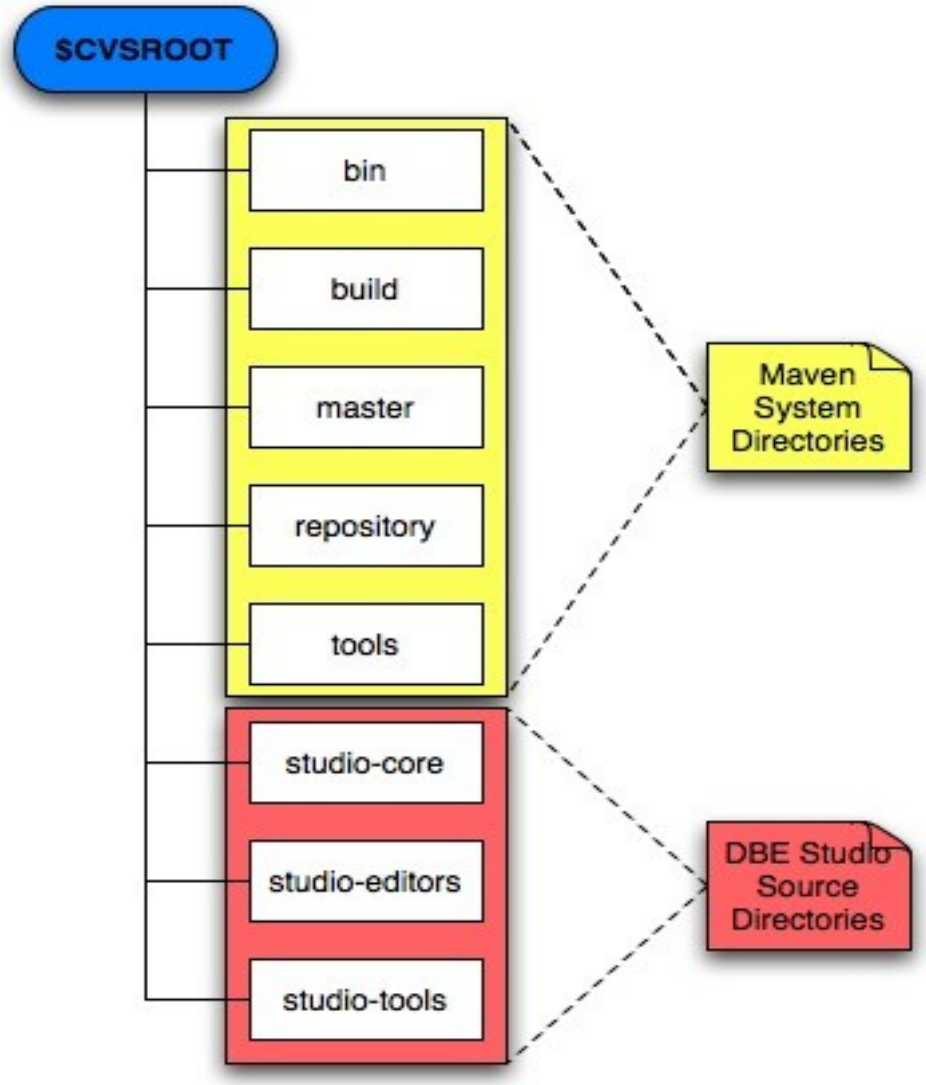


Figure 11: DBE Studio CVS Structure Overview

From the above CVS root structure the following table has been compiled. In the table, the responsible partner for implementing the named component and their role, regarding that component, is listed. In the column named "**Location**" is the location, relative to the CVS root directory where the partner's component will be committed.

DBE Studio			
Partner	**Component**	**Role**	**Location**
Soluta	SDL Editor	Implementer	`$CVSROOT/studio-editors/sdl`
	Service Manifest Creator	Implementer	`$CVSROOT/studio-tools/sm-creator`
	SDL2Java Compiler	Implementer	`$CVSROOT/studio-tools/sdl2java-compiler`
	SSL2SDL Compiler	Implementer	`$CVSROOT/studio-tools/ssl2sdl-compiler`
	SDL Storage	Implementer	`$CVSROOT/studio-tools/sdl-storage`

TCD	Manual Composer (BPEL Editor)	Implementer	`$CVSROOT/studio-editors/bpel`
	Identity Manager	Implementer	`$CVSROOT/studio-tools/identity-manager`
TUC	Ontology Editor	Implementer	`$CVSROOT/studio-editors/odm`
	BML Editor	Implementer	`$CVSROOT/studio-editors/bml`
	Ontology Viewer	Implementer	`$CVSROOT/studio-tools/ontology-viewer`
	Query Formulator	Implementer	`$CVSROOT/studio-tools/qf-sdt`
	KB-toolkit	Implementer	`$CVSROOT/studio-tools/kb-toolkit`
Intel	Studio Help	Integrator & Implementer	`$CVSROOT/studio-core/studio-help`
	Studio Example	Implementer	`$CVSROOT/studio-core/studio-examples`
	UI	Implementer	`$CVSROOT/studio-core/ui`
	Feature	Implementer	`$CVSROOT/studio-core/feature`
	Service Exporter	Implementer	`$CVSROOT/studio-tools/service-exporter`
	xServent Client	Implementer	`$CVSROOT/studio-core/servent-client`
	Perspectives	Implementer	`$CVSROOT/studio-core/perspectives`
	Preferences	Implementer	`$CVSROOT/studio-core/preferences`
FZI	User Profiling Tool	Implementer	`$CVSROOT/studio-tools/user-profiling`
UCE	BML Data Editor	Implementer	`$CVSROOT/studio-editors/bml-data`
WIT	Metering Wizard	Implementer	`$CVSROOT/studio-tools/metering-wizard`
UniS	Test Case Generator	Implementer	`$CVSROOT/studio-tools/test-case-generator`

Table 2: Responsible Partners and Components. (This table represents the cvs component structure on the DBE Studio SF project and does not directly map to the released plugins within the DBE Studio)

CVS Procedures with Eclipse

For most development tasks the preferred method to access code via CVS is through the use of the Eclipse built-in CVS feature, plugins and perspective. To access and manage CVS tasks using Eclipse follow the proceeding steps, as described below.

Eclipse CVS Configuration Pre-requisites

To guarantee correct operation with Eclipse it is necessary to configure how files are checked out of the CVS repository. In this case it means how Eclipse deals with CVS keywords. By default Eclipse will **not** expand the keywords and as such it is necessary to configure Eclipse to do this. To configure Eclipse: open the preferences page and find the preference page

shown below (Figure 12: Eclipse CVS Preferences). Ensure that the "`Default text mode`" is set to "`ASCII with keyword expansion (-kkv)`"

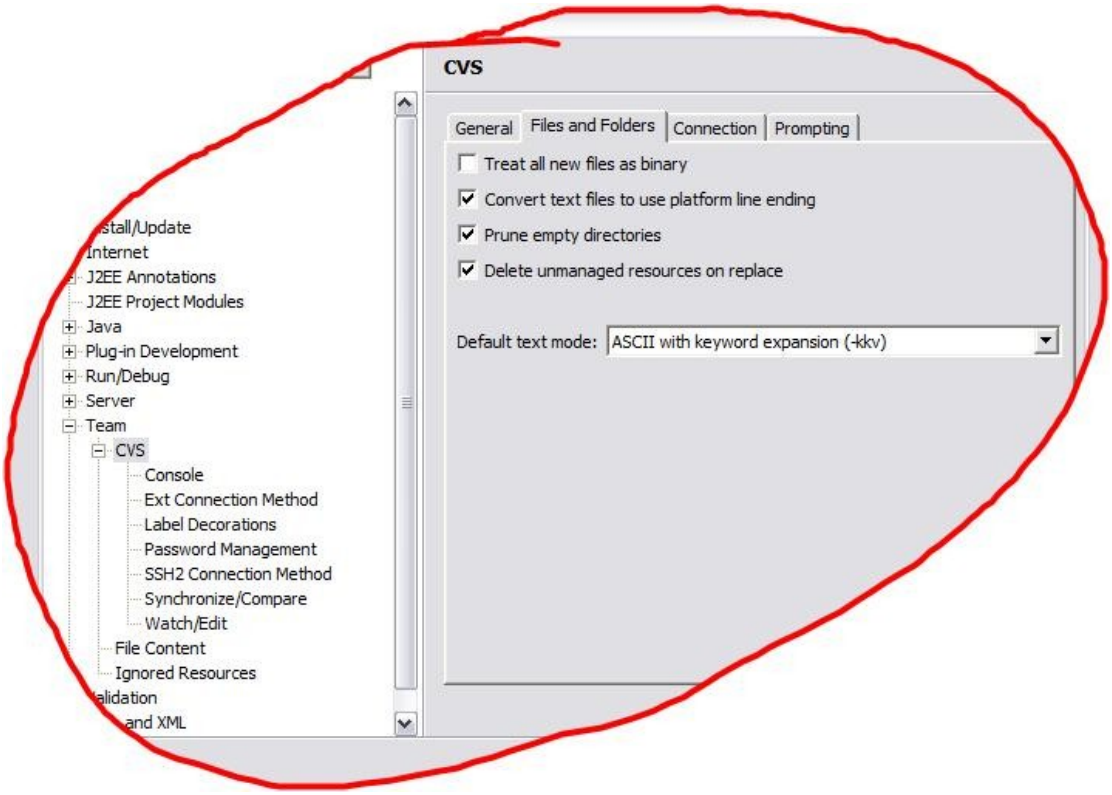


Figure 12: Eclipse CVS Preferences

CVS Workspaces

It is recommended to use separate Eclipse workspaces for CVS and plugin development tasks. This is due to the maven structure of the studio repository. The maven sub-projects, which provide the individual plugins for the studio, must be imported at the root Eclipse level for easy development. Therefore to retain the hierarchical structure of the CVS repository, we propose that developers have two workspaces, a CVS workspace and a development workspace. CVS tasks like checking out modules, branching and merging should be done in the CVS workspace and plugin development should be done in the development workspace. Workspaces can be easily switched by selecting "`File -> Switch Workspace`". This procedure with working with CVS, Eclipse and Eclipse workspaces is shown in Figure 13.

Figure 13 Overview of CVS Development Process with Workspaces

CVS Procedure Summary

This is a brief summary of basic CVS procedures within Eclipse, which are explained in more detail in the following sections.

- Switch to CVS-workspace
- Checkout the `dbestudio` Mainline (HEAD)
- Create a branch on the `dbestudio` project
- Switch to standard development-workspace
- If necessary, create the Eclipse project (e.g. on the editor or tool plugin that you are developing) by using the Maven command, "`maven eclipse`", and import into your development-workspace
- Develop and make the required changes. Note changes and new versions in the `templates\changes.xml` file within the plugin project.
- Commit to the branch as required. (Implies CVS connection must be setup inside standard development-workspace.)
- Switch to CVS-workspace when ready to merge with Mainline (Head)
- Perform CVS Merge on `dbestudio`

This process is illustrated in Figure 14.

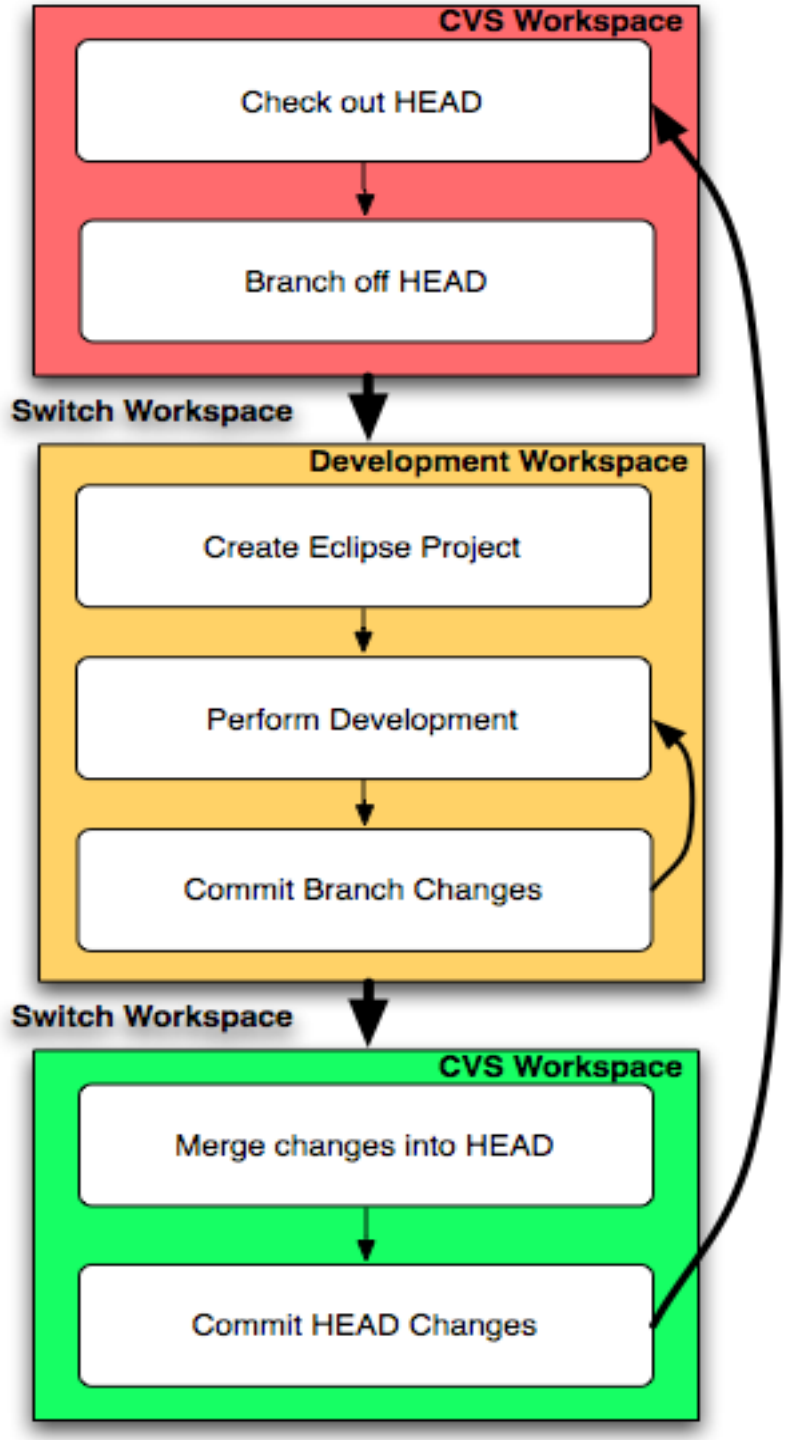


Figure 14: Detailed CVS Development Process with Workspaces

Mainline Checkouts in your CVS-workspace

If we want to check out the entire source that comprises the DBE studio the following steps will accomplish this.

1. Inside Eclipse, use "`File -> Switch Workspace`" (if necessary create it) and switch to your CVS-workspace.

2. Enter the CVS perspective within your Eclipse installation.
3. Right click anywhere in the CVS Repositories view, selecting "`New -> Repository Location…`".
4. An "`Add CVS Repository`" wizard pops up (see Figure15).
5. Here, you need to enter the host and repository path, while also including your username and password. "`extssh`" should be chosen for the connection type. Click Finish. Depending on your network / firewall connection the connection may or may not work. A SSH2 connection proxy can be defined if necessary by going to "`Window -> Preferences -> Team -> CVS -> SSH2 Connection -> Proxy`".
6. Next, enter the Resource perspective.
7. Right click, selecting "`Import -> Checkout Projects from CVS`".
8. Select the "`dbestudio`" repository location (see Figure 16), click "`Next`".
9. Select the "`dbestudio`" module (see Figure 17), click "`Next`".

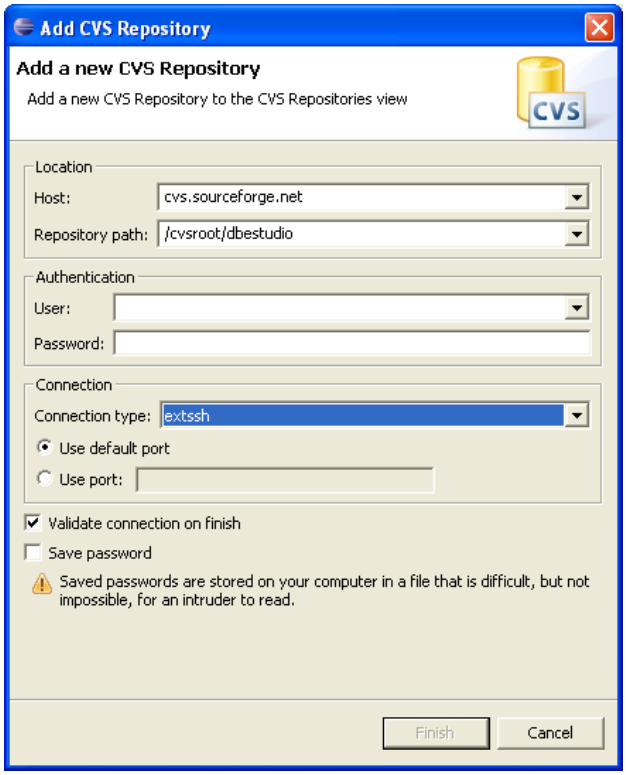


Figure 15: Add CVS Repository Wizard

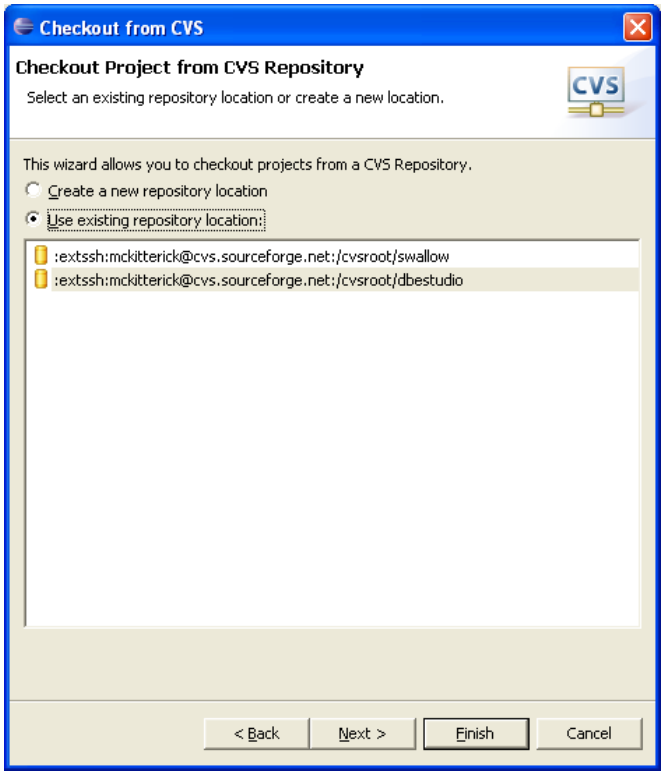


Figure 16: CVS Checkout, Selecting a Repository

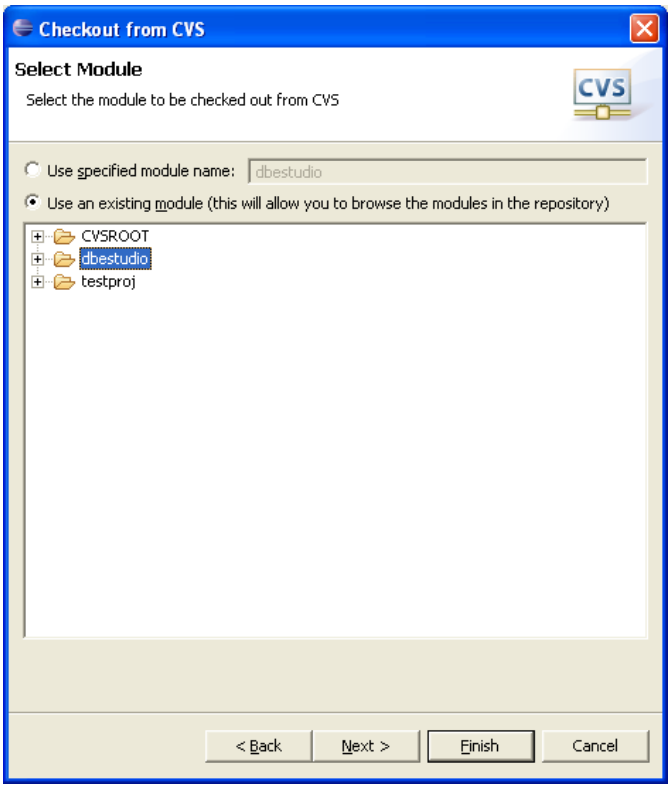


Figure 17: CVS Checkout, Selecting a Module

10. As shown in Figure 18, select "`Check out as a project in the workspace`", and then click "`Next`".

11. Select your CVS workspace location (see Figure 19), click "`Next`". It is recommended to use separate workspaces for CVS and plugin development tasks.
12. Then select "`HEAD`" for tagging, and click "`Finish`" (see Figure20).

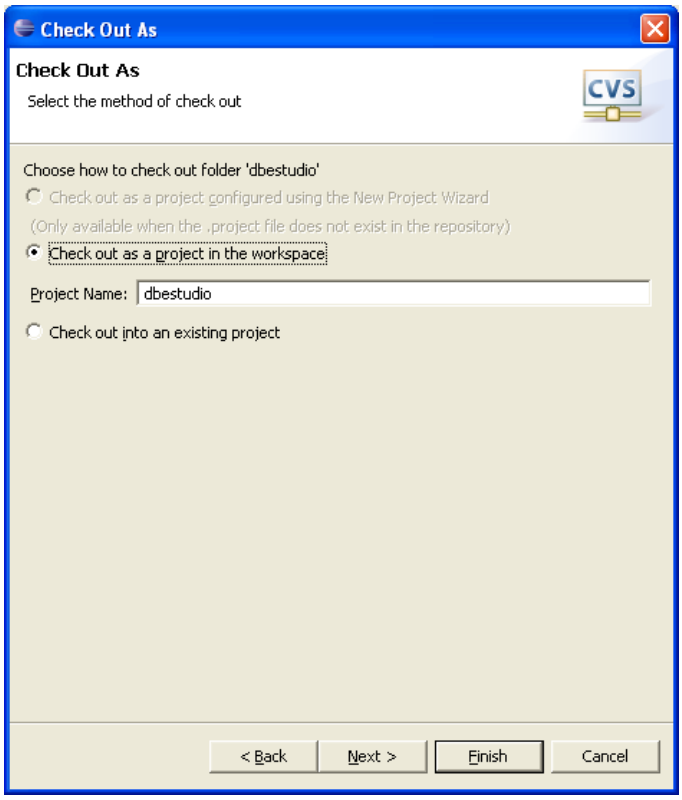


Figure 18: CVS Checkout As

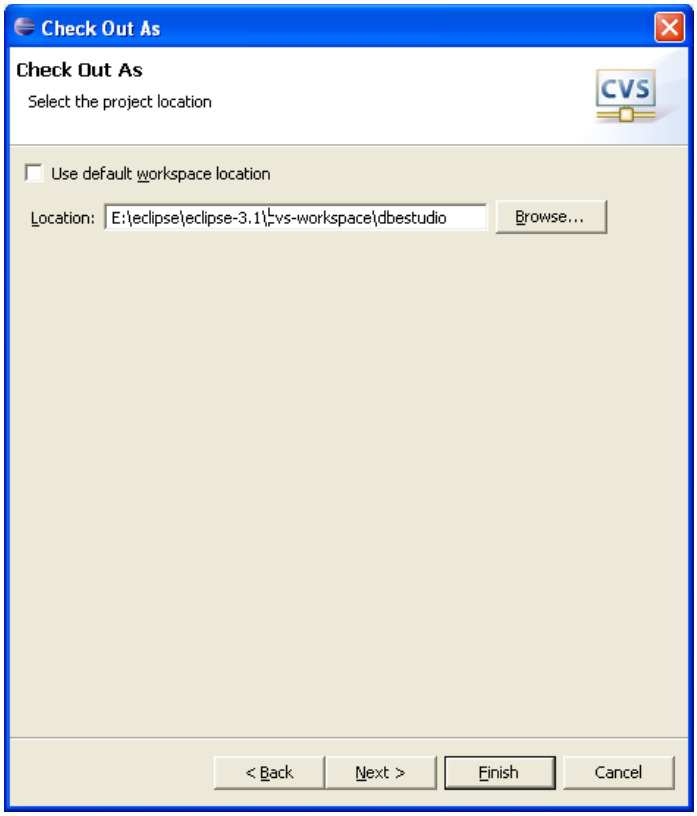


Figure 19: CVS Checkout, Select your CVS Workspace

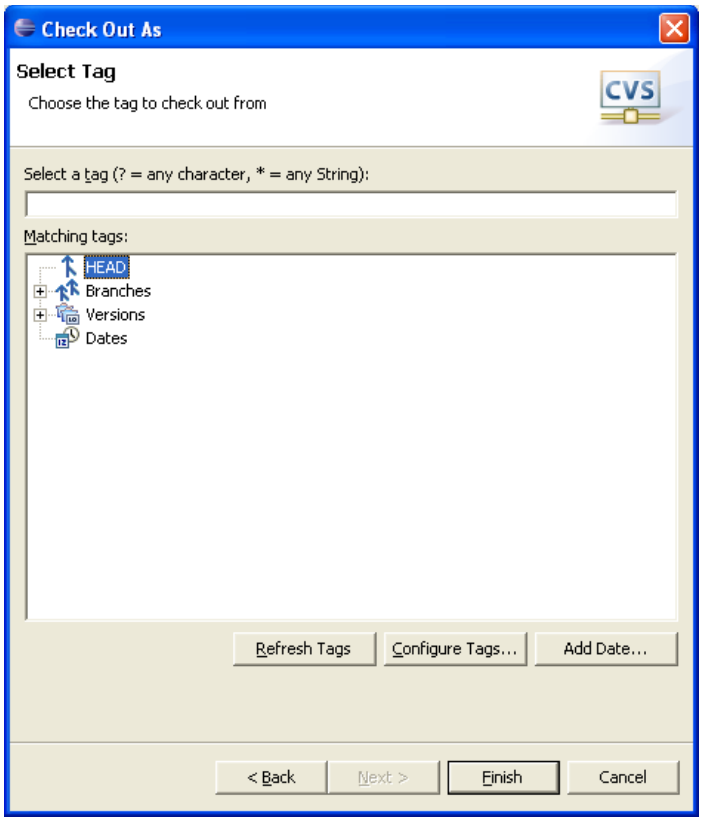


Figure 20: CVS Checkout, Select Head for tagging

Branching in your CVS-Workspace

Creating branches is done within the CVS-workspace. To create a branch off the mainline follow the proceeding steps:

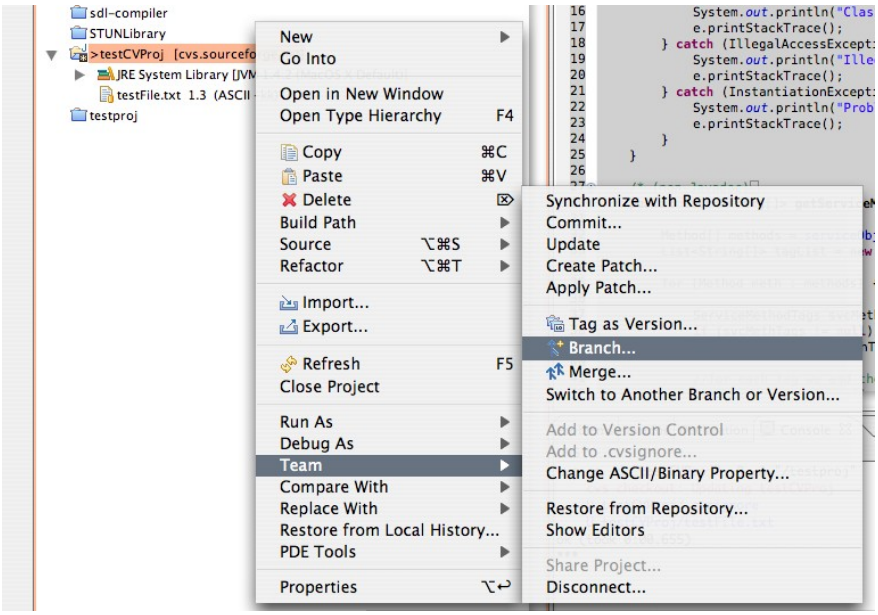


Figure 21: Branching in Eclipse (within the CVS-workspace)

Right click on the project root (Mainline HEAD) and select team and click on the entry "`Branch`" as displayed above. This will open the proceeding dialog.

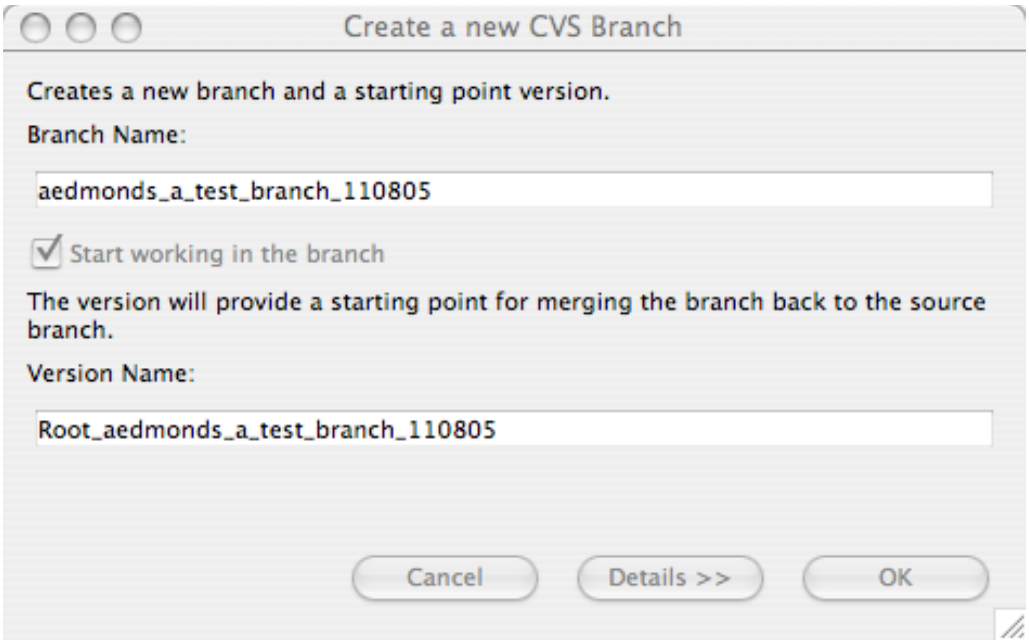


Figure 22: Naming a Branch

In this dialog, enter in the name of the branch in the format:

`<username>_<taskDescr>_<date>` e.g.
`krasik_updating_my_component_080805`

Where date is of the format `DDMMYY`.

There is no need to modify the Version name (see Figure 22). Click "`OK`" and Eclipse will begin to create a branch. Once the branching has completed, you will notice that your project root is tagged with the name of the branch (see Figure 23).

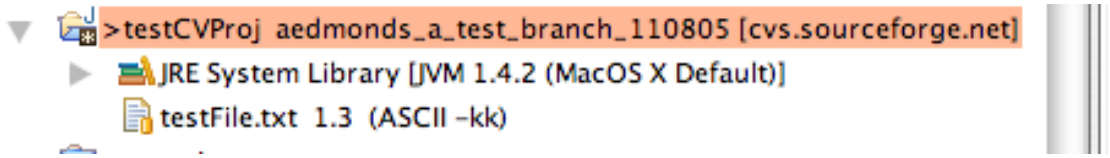


Figure 23: Eclipse Project Tagged with Branch Name

Developing in your Development-Workspace with Eclipse Project Imports

Once a branch is created, development can begin by importing an Eclipse project into your development-workspace. The following steps outline the procedure for this:

1. Use "`File -> Switch Workspace`" to switch to standard development-workspace
2. Create the Eclipse project files via Maven. E.g. on windows:
3. Open up a command prompt
4. Run `cvs-workspace\dbestudio\bin\set_env.bat` (or equivalent UNIX script)
5. Change to the directory containing the project to be worked on
6. Run "`maven eclipse`"
7. Use "`File -> Import... -> Import Existing Project into Workspace`" (i.e. development-workspace) to import the development project. Browse to the project location under the cvs-workspace directory.

Committing Changes in your Development-Workspace

Committing your changes to your branch can be done within your development-workspace, by right clicking on your project and selecting "`Team -> Commit...`" (see Figure 24).

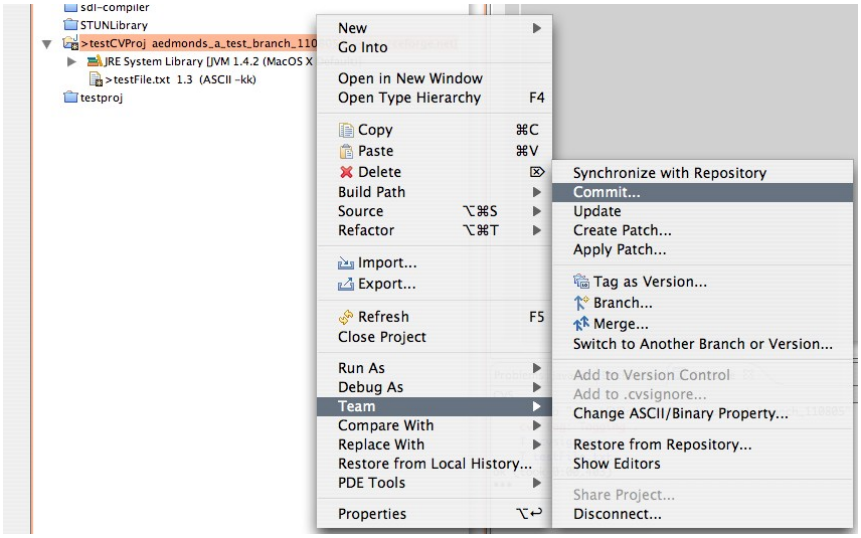


Figure 24: Committing Branched Changes in Eclipse

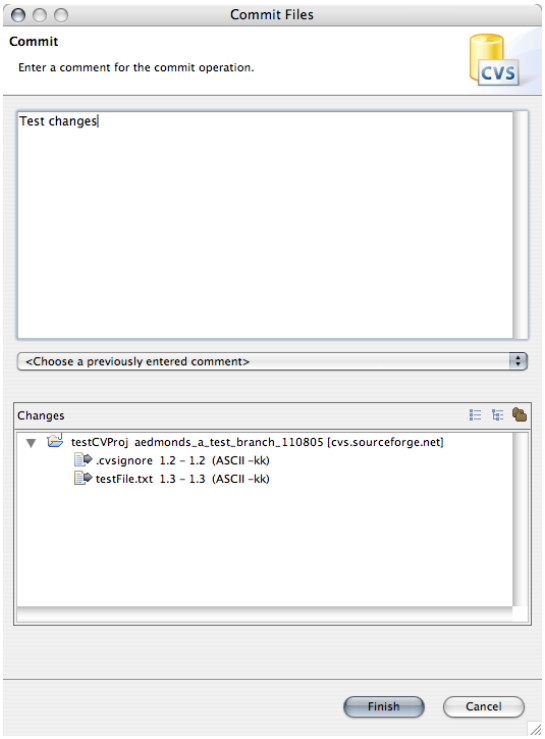


Figure 25: Commenting a Branch Commit in Eclipse

Merging your CVS Workspace

After making sure that all your code changes are committed to your branch, you must then switch back to your CVS-workspace. Now, in your CVS-workspace, it is time to merge those code changes from your branch. To do this, select "`Switch to Another Branch or Version`" from the Team menu.

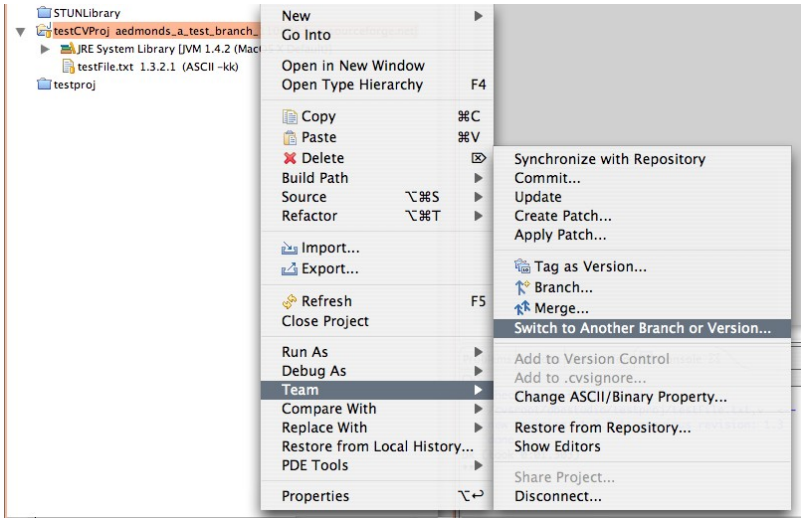


Figure 26: Switching to the Mainline for Merging

Selecting the entry now displays a dialog where you must switch back to your mainline. Select the second radio button and then the "HEAD" entry. Click "`Finish`".

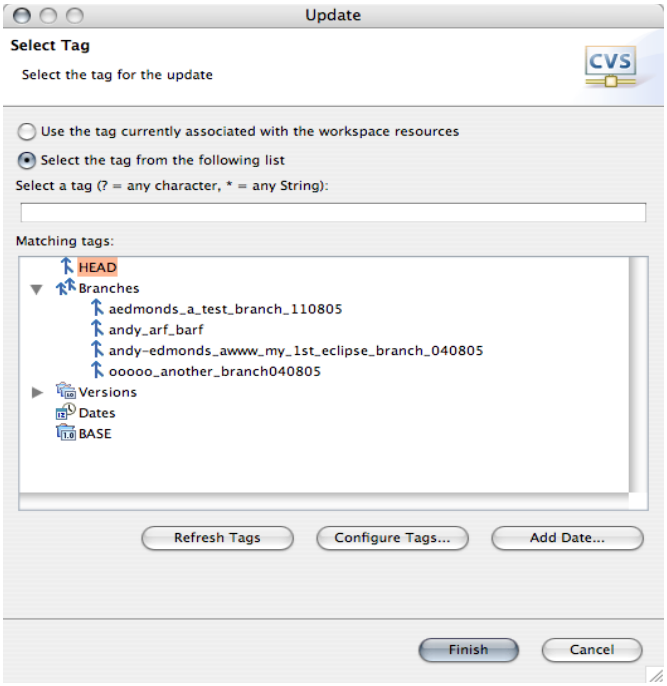


Figure 27: Selecting the Mainline

Now the actual merge is performed. Selecting the "`Merge`" entry in the Team menu does this.

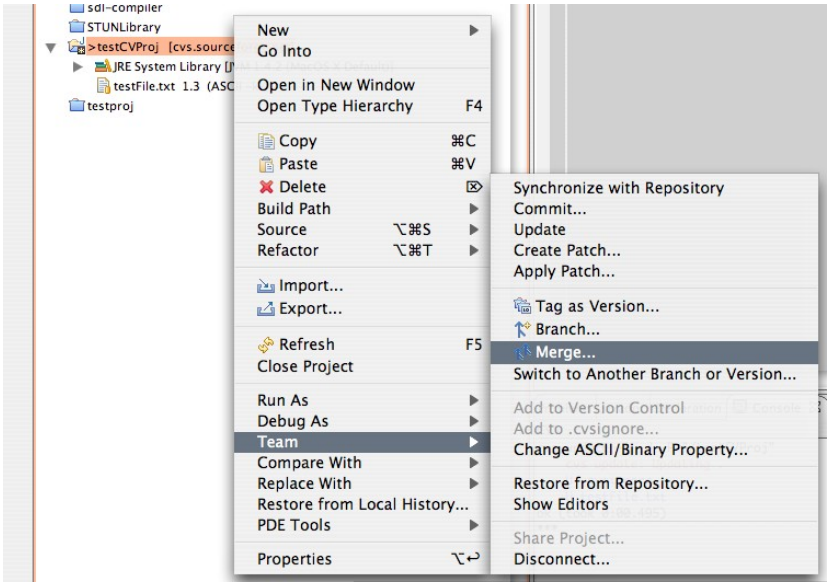


Figure 28: Initiating a Merge

Selecting "`Merge`" will display the following dialog (Figure 28), where you select the branch you want to merge into the mainline.

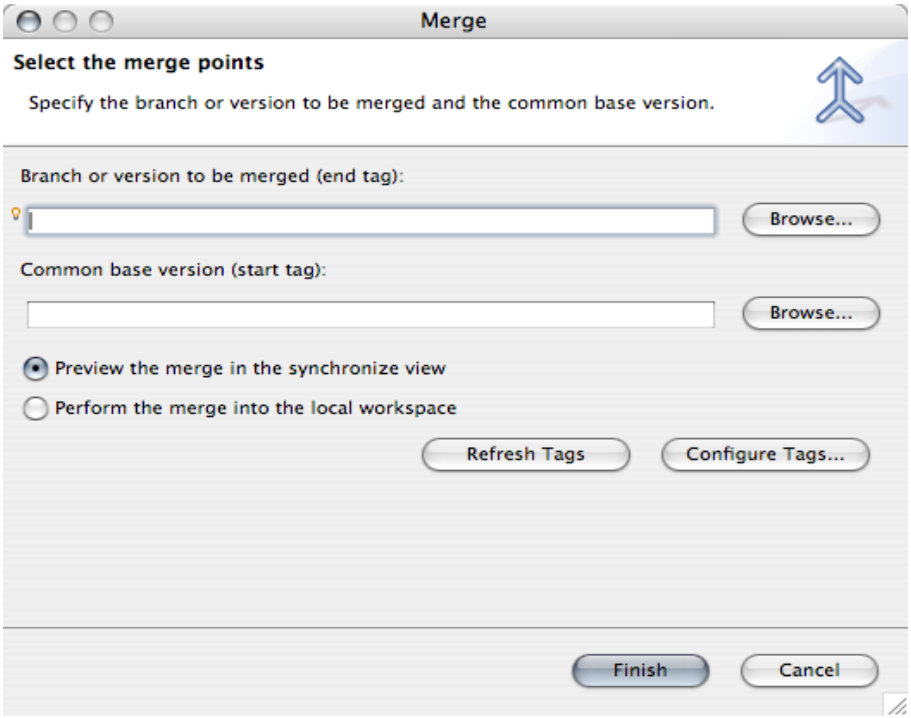


Figure 29: Merging in Eclipse

In the above dialog you need to select the branch you want to merge. To select the correct branch click on the first browse button and this will display all the branches available to merge (Figure 29). With the branch selected, select the radio button labelled "`Perform the merge into the local workspace`". Then to initiate the merge, click

"`Finish`". This will merge the changes made on the branch into your local working copy of the mainline**.** This is important to know as once merged into the local mainline you then have to commit the changes made by the merge into the CVS mainline stored on the CVS server. This is done in a similar fashion as described above (see Figure 30 – *note that when committing the branch tag will not be beside the project root as the merge is being performed on the HEAD*).

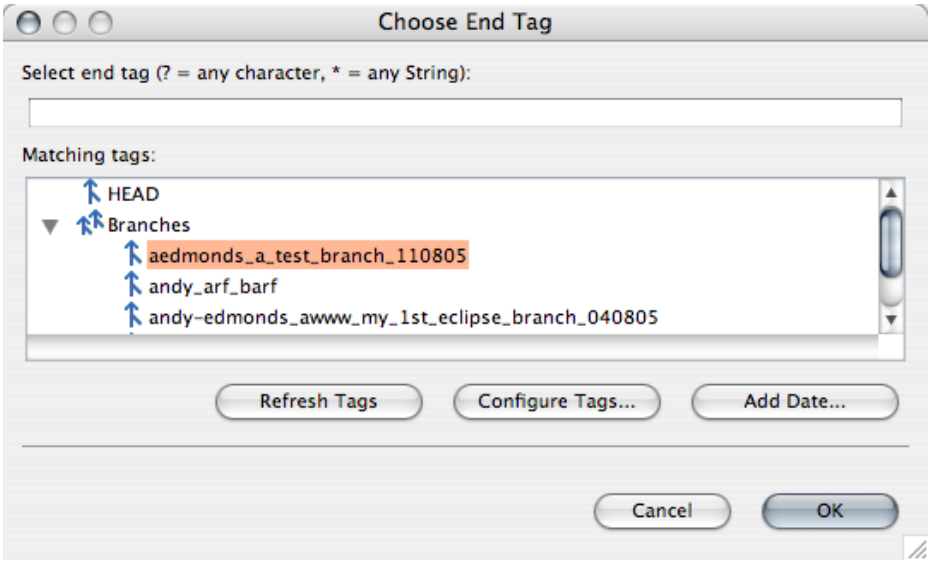


Figure 30: Selecting the Branch to Merge

Command Line Checkouts

If you wish to use the command line to interact with CVS, the following examples demonstrate some of the commands.

Mainline Checkouts

If you are a developer the following command will check out the main line (main trunk) of the DBE Studio

```
export CVS_RSH=ssh

cvs -z3 -d
:ext:$DEVELOPER_NAME@cvs.sourceforge.net:/cvsroot/dbestudio

co –P dbestudio
```

Where `$DEVELOPER_NAME` is your sourceforge login name

Anonymous Checkouts

If you only want to check out the source and do not require commit rights then the following command will suffice:

```
cvs -d
:pserver:anonymous@cvs.sourceforge.net:/cvsroot/dbestudio
login

cvs -z3 -d
:pserver:anonymous@cvs.sourceforge.net:/cvsroot/dbestudio co
-P dbestudio
```

Maven and DBE Studio

Maven [3] is the tool that creates build of DBE components and looks after any related build dependency of that component. The usage of maven and how to create a maven compliant project is detailed in [2] but for convenience the most often used maven commands are listed below:

Command	*Description*

maven jar	Creates a jar file of the component classes suitable for non-Eclipse distribution. To create an Eclipse compatible jar plug-in you need to specify the following in your maven `maven.xml` file within your project:
	
	`<goal name="jar">`
	`  <attainGoal name="eclipse-plugin:create-plugin-dist" />`
	`</goal>`

maven eclipse	Creates an Eclipse project so the maven project can be worked with in Eclipse
maven clean	Removes generated files from project

Table 3: Maven Commands

Maven Build Configuration

Maven reads the `build.properties` file that is present in your home directory. This file contains a global configuration for maven. One of the most important configuration variables is `maven.repo.remote`. This is where maven looks to see where it can find libraries to

satisfy project dependencies. If this file (`build.properties`) is not present in your home directory, you can get a copy of it from CVS and place it in your home directory. When checked out, the file can be found under:
`<CVSROOT>/dbestudio/master/src/config/build.properties`

To set the maven environment in your command line console you must run the script which can be found in `<CVSROOT>/dbestudio/bin`.

Studio Plugin and Naming Conventions

The following guidelines should be adhered to when creating or committing to CVS:

- Plugins should not specify any perspective information - this will be done by a set of perspectives for which the Integration team is responsible.
- The functionality of an editor or tool should be contained within one plugin, although a plugin may have dependencies on other sub-projects. Multiple plugins per editor/tool will just complicate the installation process.
- Source code should have a standard package structure: i.e. '`org.dbe.studio.editors.X`' or '`org.dbe.studio.tools.X`', where `X` is the name of the component a partner is responsible for.
- Plugin IDs should be named after their relevant package structure, e.g. '`org.dbe.studio.editors.sdl`'
- Plugins should follow the common menu and label naming conventions provided by Intel.

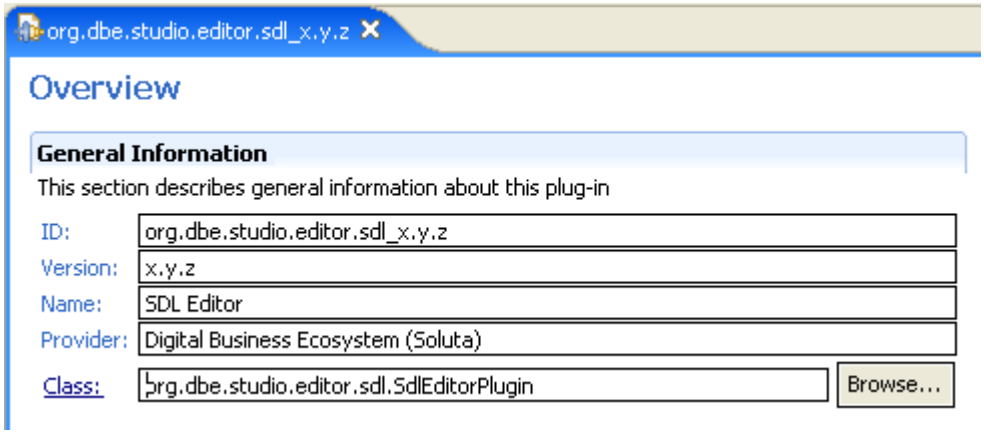


Figure 31: Example plugin.xml Overview

Versioning

All Maven sub-projects should follow a common versioning approach i.e. `x.y.z` where:

- `x` is a number denoting major milestone releases.
- `y` is a number denoting incremental feature additions to APIs and user interfaces.
- `z` is a number denoting bug fixes and minor implementation adjustments.

All component versions should follow major release and incremental feature versions of the DBE Studio. These version changes will be decided by the project integrators.

Plugin Distributions

All plugin distributions will be automatically generated by Maven. This will be accomplished by utilising the Maven Eclipse plugin, which is invoked from the '`maven jar`' command. In order to apply this functionality to your plugin and Maven sub-projects, limited modifications are required from most partners.

The example below outlines a snippet of a plugin configuration file (see Code Snippet 1), which shows the necessary changes (line `13`, shown in **bold**) to make. The plugin `id` is given a general maven artefact id will which then replace this with the project id declared in the respective maven `project.xml` file when the plugin distribution is generated. Similarly, the `version` number is also taking from the `project.xml` file. This will greatly improve the versioning, efficiency and distribution management required for a multi-partner platform distribution.

```
01 <plugin
02    name = "%pluginName"
03    id = "@maven.eclipse.plugin.artifact.id@"
04    version = "@maven.eclipse.plugin.artifact.version@"
05    provider-name = "%providerName"
06    class = "org.dbe.studio.editors.sdl.SdlEditorPlugin">
07
08    <requires>
09          ...
10    </requires>
11
12    <runtime>
13          <library name="@maven.eclipse.plugin.artifact.id@-
            @maven.eclipse.plugin.artifact.version@.jar">
14                <export name="*"/>
15          </library>
16    </runtime>
17
18 .../>
```

Code Snippet 1 Additions to `plugin.xml`

Run-time plugin dependencies can be extracted from the `project.xml` file and automatically added to the `plugin.xml` file during generation. Code Snippet 2 shows the necessary additions (lines `05-07`, shown in **bold**), which must be added to a declared dependency within a `project.xml` file. This also helps in the management of dependency version management from related sub-projects.

```
<dependency>
   <groupId>db-ecosystem</groupId>
   <artifactId>some-project-id</artifactId>
   <version>x.y.z</version>
   <properties>
         <eclipse.plugin.bundle>true</eclipse.plugin.bundle>
   </properties>
</dependency>
```

Code Snippet 2 Additions to `project.xml`

Appendix A: References

[1] SourceForge. http://www.sourceforge.net

[2] Trinity College Dublin. Dominik Dahlem. DBE Developer's Guide. Rev 0.4. May 2004.

[3] Maven build system. http://maven.apache.org

[4] Concurrent version system. http://www.gnu.org/software/cvs

[5] Eclipse project. http://www.eclipse.org