IBM Rational Team API
Tutorial for ClearCase

Legal Notices

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law.

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes

appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and

other programs (including this one) and (ii) the mutual use of the information

which has been exchanged, should contact:

IBM Corporation
Department BCFB
20 Maguire Road
Lexington, MA 02421
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available

for it are provided by IBM under terms of the IBM Customer Agreement, IBM

International Program License Agreement or any equivalent agreement between

us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurement may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has

not tested those products and cannot confirm the accuracy of performance,

compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

COPYRIGHT LICENSE:

Trademarks

Contents

Overview

This tutorial explains the basic concepts of Team API by walking you through the development of a sample application. Our sample application will make use of the ClearCase support in Team API to put a view-private file under source control in ClearCase.

We assume you are familiar with Java programming, but you have no knowledge (yet) of Team API. We also assume that you are developing your Java programs using the Eclipse IDE. (If you are unfamiliar with Eclipse, and want to obtain a copy to try, go to www.eclipse.org). Team API can also be used without Eclipse, but the Eclipse environment makes things easier. Finally, we assume some familiarity with the ClearCase Remote Client (CCRC), and that you have installed a CCRC server somewhere on your network.

Installing ClearCase Team API into Eclipse

We will begin by installing the Rational Team API into the Eclipse on your client machine, as an Eclipse "feature". This does not affect your Eclipse UI (no new perspectives or UI elements are created), but it does make the Team API libraries available in the form of Eclipse plug-ins.

Because Team API is a technical-preview release, it is present but not initially visible. You need to take the following manual step to make it visible and installable into Eclipse.

On your CCRC server machine, find the site.xml file for your CCRC Eclipse update site. On Windows servers, this will normally reside in C:\Program Files\Rational\ClearCase\web\update\site.xml. On UNIX servers, this will reside in /opt/rational/clearcase/web/update/site.xml. You will need to enable the "Rational TeamAPI" feature in your site.xml file to make Team API visible as an Eclipse feature. The entry for this feature is already present in the site.xml file. To enable it, find the lines that say "DELETE THIS LINE TO ENABLE CC TeamAPI feature" and delete them.

Installing the Rational Team API feature

To find and install the Rational Team API into your Eclipse environment:

1. Under the Eclipse **Help** menu**,** select **Software Updates -> Find and Install…** This brings up the Eclipse Update Manager. The following sections walk you through the process of using the Eclipse Update Manager to install Team API.
2. In the **Install** window, for **Feature Updates,** select **Search for new features to install**. Click **Next**.
3. In the **Update sites to visit** window, click **New Remote Site** to define the update site from which to download the IBM Rational Team API.
4. In the **New Update Site** window, enter a Name and a URL to define your site, then click OK. You can give the site any name you want (for example, "CCRC and TeamAPI"). In the URL field, you point to the URL of your CCRC server machine's CCRC update site. Typically this URL is "http://<*your-machine*>/ccrc/update." Click **OK**.
5. Your update site then appears in the choice list in the **Update sites to visit** window. Select your site (for example CCRC and TeamAPI). Click **Next**.
6. The **Search Results** window shows the features available to install at this site. Since ClearCase Team API is sharing an update site with ClearCase Remote Client, both show up as features you can install. You can check one or the other or both. In this example, we will only check the Team API feature. Click **Next**.
7. In the **Feature License** window, accept the license agreement. Click **Next**.
8. In the **Install Location** window, click **Finish.**
9. In the **Jar Verification** window, click **Install**.

You have now installed the Rational Team API into Eclipse, and will see three new directories appear under your Eclipse plugins/ directory:

com.ibm.rational.teamapi.core_7.0.1

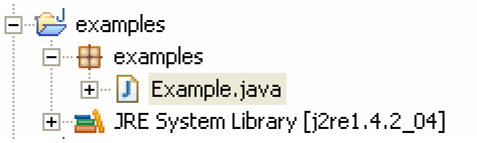
com.ibm.rational.teamapi.cc_7.0.1

com.ibm.rational.clearcase.core_7.0.1

We'll see in subsequent sections how to make use of the contents of these directories.

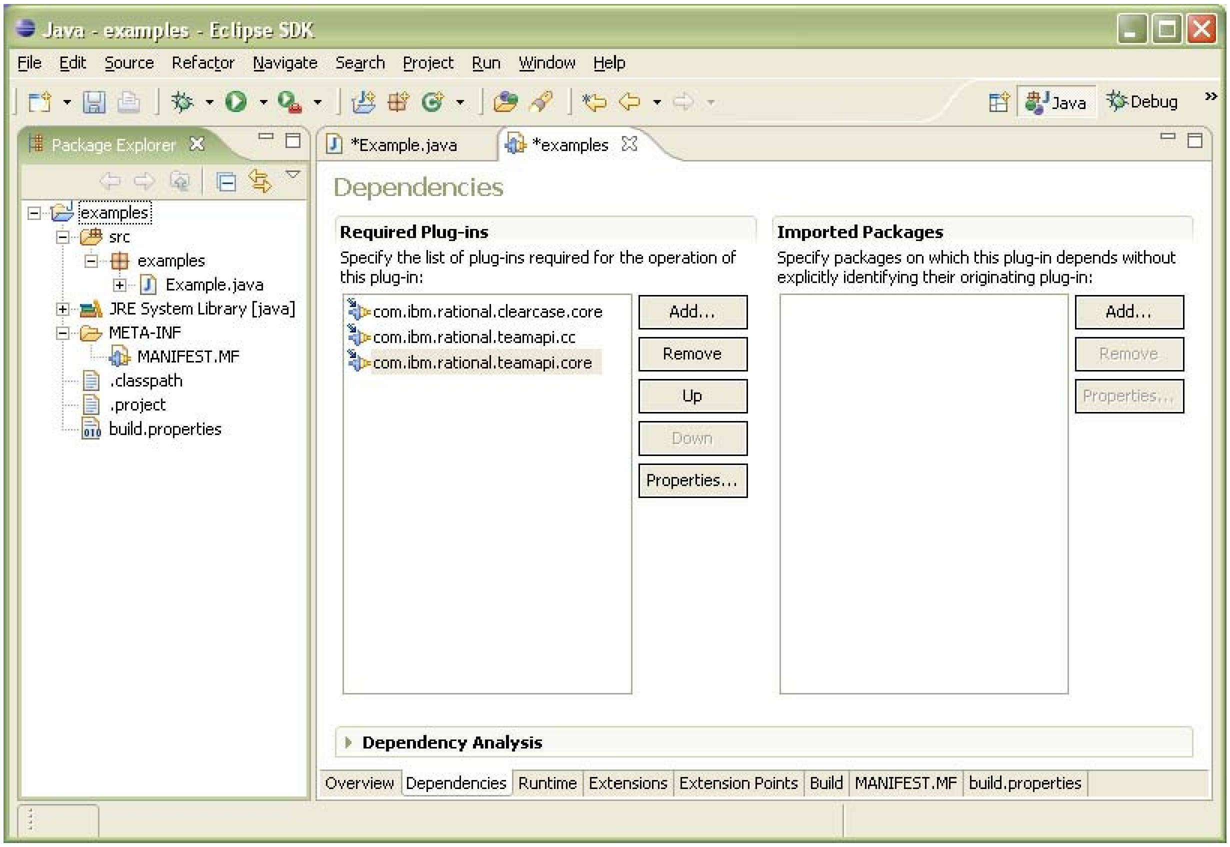
Creating a project to house the example program

Next we will create a new Eclipse project. The most straightforward way to do that is "File->New->Project->Java Project" followed by "File->New->Package" followed by "File->New->Class". Here we'll select "examples" as both the project name and the package name, and "Example" as the Java class name, resulting in a project structure like this:

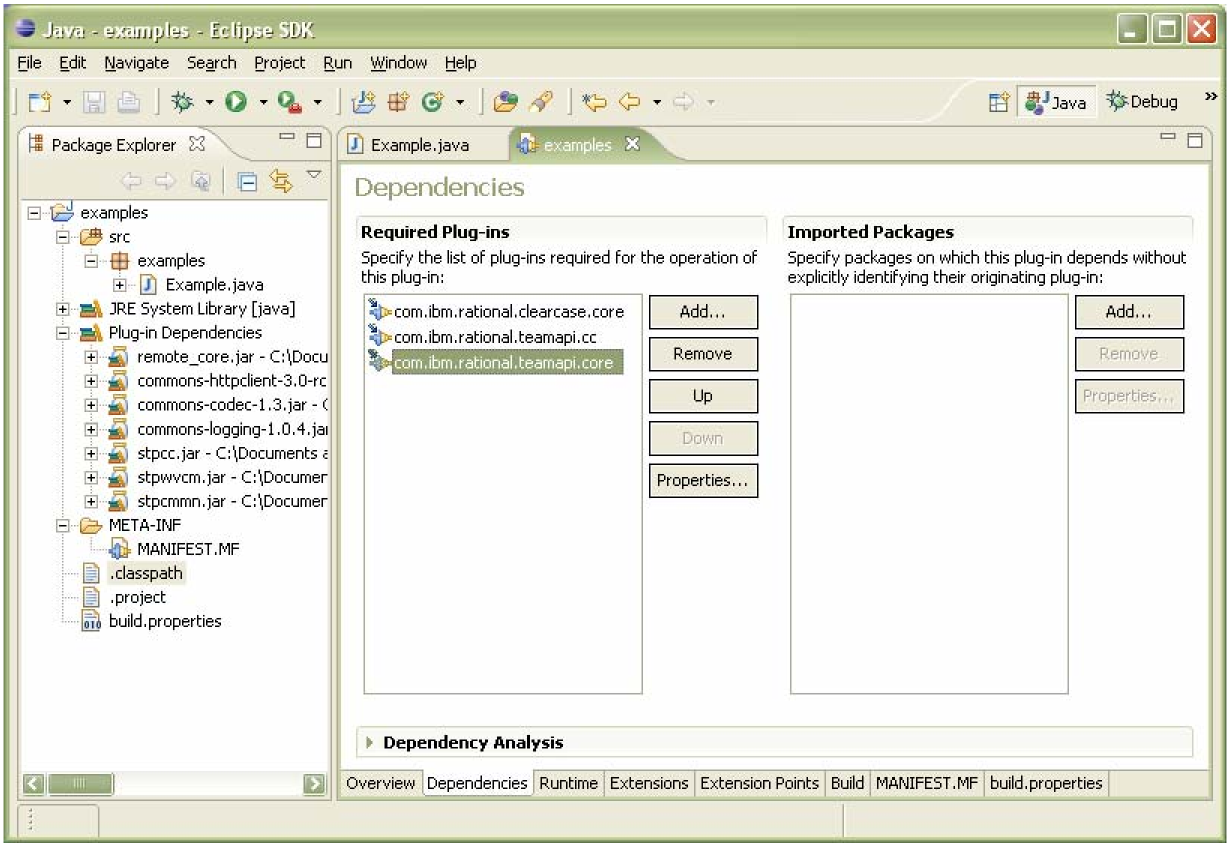


Next we will play a little trick and turn the project into an Eclipse plug-in. The reason we do this is to make use of Eclipse mechanisms for adding the Team API JAR's to the classpath. By making our "examples" project a plugin, we can declare that it depends on the Team API plugins, and then Eclipse automatically adds the Team API jars to the classpath. To convert this project to a plug-in:

1. Click right on the "examples" project to bring up the project context menu. Select "PDE Tools->Convert Projects to Plug-in Projects…". Select the "examples" project and click the "Finish" button.
2. In the Project Explorer pane, you will see a new directory "META-INF" under the "examples" project. Open this directory and double-click on the "MANIFEST.MF" file, which will open the project manifest in an editing window.
3. Click on the "Dependencies" tab at the bottom of the manifest editor:
4. Click on the "Add Required Plug-ins…" button and add the three TeamAPI plug-ins:
   com.ibm.rational.teamapi.core_7.0.1
   com.ibm.rational.teamapi.cc_7.0.1
   com.ibm.rational.clearcase.core_7.0.1
   When you finish this step, the project manifest should list those plug-ins in the **Required Plug-ins** pane:


Java - examples - Eclipse SDK
File Edit Source Refactor Navigate Search Project Run Window Help
Java
Debug
Package Explorer
examples
src
examples
Example.java
JRE System Library [java]
META-INF
MANIFEST.MF
.classpath
.project
build.properties
*Example.java
*examples
Dependencies
Required Plug-ins
Specify the list of plug-ins required for the operation of this plug-in:
com.ibm.rational.clearcase.core
com.ibm.rational.teamapi.cc
com.ibm.rational.teamapi.core
Add...
Remove
Up
Down
Properties...
Imported Packages
Specify packages on which this plug-in depends without explicitly identifying their originating plug-in:
Add...
Remove
Properties...
Dependency Analysis
Overview
Dependencies
Runtime
Extensions
Extension Points
Build
MANIFEST.MF
build.properties

5. Save these changes, then bring up the “examples” project’s context menu and select “PDE Tools->Update Classpath…” to add the TeamAPI plug-in jars to your project’s classpath. If you open up the “examples” project’s “Plug-in Dependencies” folder, you should now see the TeamAPI jars:



Required Plug-ins

Specify the list of plug-ins required for the operation of this plug-in:

- com.ibm.rational.clearcase.core
- com.ibm.rational.teamapi.cc
- com.ibm.rational.teamapi.core

Imported Packages

Specify packages on which this plug-in depends without explicitly identifying their originating plug-in:

Example.java: Framework

We'll start with the main method of our sample Java application because it shows the overall structure of the application. The full source code is in Appendix A, and can be copied directly into the Example.java file you just created in Eclipse.

```
public static void main(String[] args) {

 trace("Processing program arguments");
 if (args.length != 4) {
    printUsageMsg();
    return;
 }
 String ccrcServerUrl = args[0];
 String login = args[1];
 String password = args[2];
 File testFile = new File(args[3]);

 try {
    trace("Instantiating ClearCase TeamAPI provider");
    CcProvider provider = getProvider(ccrcServerUrl, login, password);

    trace("Constructing proxy for test file");
    ControllableResource testRes = getProxy(testFile, provider);
```

```
            trace("Constructing proxy for test file's parent directory");
            File parentDir = testFile.getParentFile();
            ControllableResource parent = getProxy(parentDir, provider);


            trace("Adding test file to source control: " + testFile);
            addToSourceControl(testRes, parent);

            trace("Printing test file's properties");
            printProps(testRes);

            trace("Done");

        } catch (Exception e) {
            trace("ClearCase TeamAPI example failed.  Printing stack trace.");
            e.printStackTrace();
        }
            }
```

What Packages to Import

Our example imports packages from several sources: WVCM (javax.wvcm.*), the TeamAPI base extensions to WVCM (com.ibm.rational.wvcm.stp.*), and the ClearCase-specific Team API extensions (com.ibm.rational.wvcm.stp.cc.*). Here are the **import** statements for our example application:

```
import java.io.File;

import javax.wvcm.ControllableFolder;
import javax.wvcm.ControllableResource;
import javax.wvcm.PropertyNameList;
import javax.wvcm.ProviderFactory;
import javax.wvcm.Version;
import javax.wvcm.WvcmException;
import javax.wvcm.PropertyNameList.PropertyName;
import javax.wvcm.ProviderFactory.Callback;

import com.ibm.rational.wvcm.stp.StpProvider;
import com.ibm.rational.wvcm.stp.cc.CcProvider;
```

Getting a provider object

The first thing a Rational Team API application needs to do is get a provider object. The provider ties the Rational Team API interface classes to the ClearCase implementation. Here is our code for getting a provider object:

```
/**
    * Get the ClearCase TeamAPI provider for this example.
    * Establish authentication callback for logging in to the CCRC server.
    * @return new ClearCase provider
    * @throws Exception if provider cannot be instantiated
    */
   static CcProvider getProvider(
         String serverUrl, String login, String password) throws Exception
```

```
{
   Callback callback = new MyAuthCallback(serverUrl, login, password);
   CcProvider provider = (CcProvider) ProviderFactory.createProvider(
         CcProvider.CLIENT_HOSTED_PROVIDER,
         callback);
   provider.setDefaultServerUrl(StpProvider.Domain.CLEAR_CASE, serverUrl);

   return provider;
}
```

`Provider.CLIENT_HOSTED_CLEARCASE_PROVIDER` is a Rational Team API string constant that names our provider implementation class ("com.ibm.rational.stp.client.internal.core.ClientHostedProvider").

The call to `provider.setDefaultServerUrl()` provides a server context for operations that don't otherwise have one.

The MyAuthCallback class implements the WVCM ProviderFactory's authentication callback interface. The Rational Team API provider will invoke this callback to obtain your credentials for logging in to the ClearCase CCRC server. Here is the MyAuthCallback implementation:

```
/**
    * TeamAPI Authentication callback. This callback is invoked when the
    * ClearCase TeamAPI provider logs in to the CCRC server.
    */
   private static class MyAuthCallback implements Callback {

      private final String serverUrl;
      private final String login;
      private final String password;

      public MyAuthCallback(String serverUrl, String login, String password) {
         this.serverUrl = serverUrl;
         this.login = login;
         this.password = password;
      }

      public Authentication getAuthentication(String realm, int retries) {
         trace("Logging in to CCRC server: " + serverUrl);
         trace("Login:    " + login);
         trace("Password: " + password);

         // Retry count > 0 means authentication failed.
         // Perhaps password was wrong or login domain was left out?
         myAssert(retries == 0);

         return new Authentication() {
            public String loginName() {
               // On Windows servers, this should return domain and
               // username separated by a backslash
               return login;
            }
            public String password() {
               return password;
            }
        };
    }
}
```

Proxies and locations

Now that we have a provider, what do we do with it? The main function of the provider is to hand out proxies. A proxy is an object in memory representing a persistent resource on the client or server (for example, a file in a ClearCase view, a version of that file, or a UCM activity). Proxies are used to retrieve information about the resource or perform operations on the resource. In this example, we need a proxy to the file we're checking out.

Some explanation of the Team API object model is required before we go further. The base class of any proxy object is Resource. Every proxy is a Resource or an extension of a Resource. All Resources have a Location, some properties, and possibly content. A Location is simply an object containing a string (e.g. a path) saying where the persistent resource lives on the client or server. Properties will be discussed in more detail later. Content is just a byte stream (e.g., a file's content).

A ControllableResource is an extension to Resource that represents a file that is or can be put under source control, i.e. a file in a view. This will be the type of our proxy. Here is our sample application's method for creating a ControllableResource proxy from a Java file instance:

```
/**
 * Construct a ControllableResource proxy for the specified file.
 */
static ControllableResource getProxy(File file, CcProvider provider)
   throws WvcmException
{
   return provider.controllableResource
      (provider.filePathLocation(StpProvider.Domain.CLEAR_CASE, file));
}
```

The file path location is used to create a WVCM location, which is then passed to the ClearCase TeamAPI provider to create a ControllableResource proxy for our file.

The provider contains similar methods to create all types of proxies. For example,provider.folder(Location) creates a Folder proxy, provider.workspace(Location) creates a Workspace proxy, provider.version(Location) creates a Version proxy, and provider.ccActivity(Location) creates a UCM activity proxy.

Properties

All resources have properties. Properties describe nearly everything about the resource. Some examples of properties available to a Resource (and hence all its subclasses) are CREATION_DATE, DISPLAY_NAME, and RESOURCE_IDENTIFIER. There are also properties available on specific subclasses. For example, a ControllableResource (file in a view) has a IS_CHECKED_OUT property saying if the file is checked out or not.

Properties can be read and written to/from the proxy via the Resource.getXxx() and Resource.setXxx(...) methods, where Xxx represents a property name. However, properties cannot be read from the proxy until they are read from the persistent resource first. This is done by calling doReadProperties(PropertyNameList) on the resource. Similarly, properties set in the proxy are not applied to the persistent resource until Resource.doWriteProperties() is called. In general, it is important to remember that getXxx() methods return values of properties from the proxy, setXxx() methods set values of properties in the proxy, and doXxx() methods operate on the persistent resource and have no effect on the proxy.

The doReadProperties() method is designed to let you fetch multiple property values in a single call, thus minimizing the number of client/server RPCs. Further, this method supports fetching *nested* properties – properties of properties. For instance, if you have a ControllableResource proxy, you can ask for it's "CHECKED_IN" property. This returns a Version proxy for that file's checked in version. At the same time, you can fetch properties of that checked in version. In the `printProps()` method below, we do just that, fetching several properties of the file, including its CHECKED_IN version, *plus* the checked in version's CREATION_DATE and VERSION_NAME properties. All in a single call to doReadProperties().

```
/**
  * Print some interesting properties of the specified resource.
  */
 static void printProps(ControllableResource res) throws WvcmException {

    PropertyNameList nestedProps = new PropertyNameList(
       new PropertyName[] {
          ControllableResource.IS_VERSION_CONTROLLED,
          ControllableResource.IS_CHECKED_OUT,
             ControllableResource.CHECKED_IN.nest(
                new PropertyName[] {
                   Version.CREATION_DATE,
                   Version.VERSION_NAME})});

    res = (ControllableResource) res.doReadProperties(nestedProps, null);

    trace("Resource is controlled:  " +
          Boolean.toString(res.getIsVersionControlled()));
    trace("Resource is checked out: " +
          Boolean.toString(res.getIsCheckedOut()));

    Version newVersion = res.getCheckedIn();

    trace("Resource's checked in version: " + newVersion.getVersionName());
    trace("Checked in version created on: " + newVersion.getCreationDate());
 }
```

Performing operations

Ultimately, your application probably wants to *do* something, that is, perform operations on the underlying resources. You perform operations by calling methods on the proxies. Methods that do something start with "do" and are of the form do*Something()*. Here are some examples of operations you can perform using a ControllableResource proxy "cr". These methods all have a simple form as well as forms allowing you to specify various arguments:

cr.doCreateResource() – creates the underlying file on the file-system

cr.doControl() – puts the resource under version control; equivalent to ClearCase "mkelem"

cr.doCheckout() – checks out the file

cr.doUncheckout() – un-does the checkout

cr.doCheckin() – checks in the file, thus creating a new version and adding that to the version history

In our example application, the `addToSourceControl()` method uses several “do” methods to accomplish its task, including `doCheckout()` and `doCheckin()` to check out and check in the parent directory, and `doControl()` to actually add the specified file to source control:

```
/**
 * Put the specified file under ClearCase source control. The file must
 * reside in a ClearCase web view in a source-controlled directory,
 * but must not already be under source control itself.
 */
static void addToSourceControl(
        ControllableResource res,
        ControllableResource parent) throws Exception
{
    PropertyNameList wantedProps = new PropertyNameList(
            new PropertyName[] {
                    ControllableResource.IS_CHECKED_OUT,
                    ControllableResource.IS_VERSION_CONTROLLED });

    trace("Verifying that file is NOT already source controlled...");
    res = (ControllableResource) res.doReadProperties(wantedProps, null);
    myAssert(res.getIsVersionControlled() == false);

    trace("Verifying that the parent folder IS source controlled...");
    parent = (ControllableFolder) parent.doReadProperties(wantedProps, null);
    myAssert(parent.getIsVersionControlled() == true);

    if (parent.getIsCheckedOut()) {
        trace("Parent folder is already checked out");
    } else {
        // In order to add the test file to source control,
        // the parent directory must first be checked out.
        trace("Checking out parent folder...");
        parent.doCheckout(null);
    }

    trace("Adding file to source control...");
    res.doVersionControl(null);

    trace("Checking in parent folder...");
    parent.doCheckin(null);
}
```

Appendix A: Example Program Code

Here is the full source code of Example.java. Copy this and paste it into the Example.java class file in your Eclipse project:

```
package example;

import java.io.File;

import javax.wvcm.ControllableFolder;
import javax.wvcm.ControllableResource;
import javax.wvcm.PropertyNameList;
import javax.wvcm.ProviderFactory;
import javax.wvcm.Version;
import javax.wvcm.WvcmException;
import javax.wvcm.PropertyNameList.PropertyName;
import javax.wvcm.ProviderFactory.Callback;

import com.ibm.rational.wvcm.stp.StpProvider;
import com.ibm.rational.wvcm.stp.cc.CcProvider;

/**
 * This sample application uses the ClearCase TeamAPI to
 * put a file under source control.
 */
public class Example {

   private static boolean tracing = true;

   /**
    * TeamAPI Authentication callback. This callback is invoked when the
    * ClearCase TeamAPI provider logs in to the CCRC server.
    */
   private static class MyAuthCallback implements Callback {

      private final String serverUrl;
      private final String login;
      private final String password;

      public MyAuthCallback(String serverUrl, String login, String password) {
         this.serverUrl = serverUrl;
         this.login = login;
         this.password = password;
      }

      public Authentication getAuthentication(String realm, int retries) {
         trace("Logging in to CCRC server: " + serverUrl);
         trace("Login:    " + login);
         trace("Password: " + password);

         // Retry count > 0 means authentication failed.
         // Perhaps password was wrong or login domain was left out?
         myAssert(retries == 0);

         return new Authentication() {
            public String loginName() {
               // On Windows servers, this should return domain and
               // username separated by a backslash
               return login;
            }
            public String password() {
               return password;
            }
         };
    }
}

static void printUsageMsg() {
    System.out.println("This program expects the following arguments:");
    System.out.println("1) URL of a ClearCase CCRC server");
    System.out.println("2) CCRC server login");
    System.out.println("   If the CCRC server is a Windows system,");
    System.out.println("   specify both the domain and login");
    System.out.println("   separated by a backslash: domain\\login.");
    System.out.println("3) CCRC server password");
    System.out.println("4) Test file to add to source control");
    System.out.println("   This file must reside in a loaded, source-");
    System.out.println("   controlled directory in a ClearCase");
    System.out.println("   web view on the local machine.");
    System.out.println();
    System.out.println("For example:");
    System.out.println();
    System.out.println("  http://my_server/ccrc  my_domain\\my_login");
    System.out.println("        my_password  C:/webviews/a/b/c.txt");
}

static void myAssert(boolean expr) {
    if ( ! expr) {
        throw new AssertionError();
    }
}

static void trace(String string) {
    if (tracing) {
        System.out.println("##### " + string);
    }
}

/**
 * Get the ClearCase TeamAPI provider for this example.
 * Establish authentication callback for logging in to the CCRC server.
 * @return new ClearCase provider
 * @throws Exception if provider cannot be instantiated
 */
static CcProvider getProvider(
        String serverUrl, String login, String password) throws Exception
{
    Callback callback = new MyAuthCallback(serverUrl, login, password);
    CcProvider provider = (CcProvider) ProviderFactory.createProvider(
            CcProvider.CLIENT_HOSTED_PROVIDER,
            callback);
    provider.setDefaultServerUrl(StpProvider.Domain.CLEAR_CASE, serverUrl);

    return provider;
}

/**
 * Construct a ControllableResource proxy for the specified file.
 */
static ControllableResource getProxy(File file, CcProvider provider)
   throws WvcmException
{
   return provider.controllableResource
      (provider.filePathLocation(StpProvider.Domain.CLEAR_CASE, file));
}

/**
 * Print some interesting properties of the specified resource.
 */
static void printProps(ControllableResource res) throws WvcmException {

   PropertyNameList nestedProps = new PropertyNameList(
      new PropertyName[] {
         ControllableResource.IS_VERSION_CONTROLLED,
         ControllableResource.IS_CHECKED_OUT,
            ControllableResource.CHECKED_IN.nest(
               new PropertyName[] {
                  Version.CREATION_DATE,
                  Version.VERSION_NAME})});

   res = (ControllableResource) res.doReadProperties(nestedProps, null);

   trace("Resource is controlled:  " +
         Boolean.toString(res.getIsVersionControlled()));
   trace("Resource is checked out: " +
         Boolean.toString(res.getIsCheckedOut()));

   Version newVersion = res.getCheckedIn();

   trace("Resource's checked in version: " + newVersion.getVersionName());
   trace("Checked in version created on: " + newVersion.getCreationDate());
}

/**
 * Put the specified file under ClearCase source control. The file must
 * reside in a ClearCase web view in a source-controlled directory,
 * but must not already be under source control itself.
 */
static void addToSourceControl(
      ControllableResource res,
      ControllableResource parent) throws Exception
{
   PropertyNameList wantedProps = new PropertyNameList(
         new PropertyName[] {
               ControllableResource.IS_CHECKED_OUT,
               ControllableResource.IS_VERSION_CONTROLLED });

   trace("Verifying that file is NOT already source controlled...");
   res = (ControllableResource) res.doReadProperties(wantedProps, null);
   myAssert(res.getIsVersionControlled() == false);

   trace("Verifying that the parent folder IS source controlled...");
```

```
        parent = (ControllableFolder) parent.doReadProperties(wantedProps, null);
        myAssert(parent.getIsVersionControlled() == true);

        if (parent.getIsCheckedOut()) {
            trace("Parent folder is already checked out");
        } else {
            // In order to add the test file to source control,
            // the parent directory must first be checked out.
            trace("Checking out parent folder...");
            parent.doCheckout(null);
        }

        trace("Adding file to source control...");
        res.doVersionControl(null);

        trace("Checking in parent folder...");
        parent.doCheckin(null);
    }

    public static void main(String[] args) {

        trace("Processing program arguments");
        if (args.length != 4) {
            printUsageMsg();
            return;
        }
        String ccrcServerUrl = args[0];
        String login = args[1];
        String password = args[2];
        File testFile = new File(args[3]);

        try {
            trace("Instantiating ClearCase TeamAPI provider");
            CcProvider provider = getProvider(ccrcServerUrl, login, password);

            trace("Constructing proxy for test file");
            ControllableResource testRes = getProxy(testFile, provider);

            trace("Constructing proxy for test file's parent directory");
            File parentDir = testFile.getParentFile();
            ControllableResource parent = getProxy(parentDir, provider);

            trace("Adding test file to source control: " + testFile);
            addToSourceControl(testRes, parent);

            trace("Printing test file's properties");
            printProps(testRes);

            trace("Done");

        } catch (Exception e) {
            trace("ClearCase TeamAPI example failed.  Printing stack trace.");
            e.printStackTrace();
        }
    }
}
```

Executing the Example Application

It is very easy to execute our example application from Eclipse, assuming you have a working CCRC environment. The sample application takes several command line arguments, described below. For more information about these command line arguments, run the example application with no arguments and it will print a helpful usage message.

To run the example application in Eclipse, click right on the Example.java class in the Package Explorer view, and select "Run as->Run . . .". In the "Create, manage, and run" dialog that comes up, click right on the "Java Application" node in the "Configurations" browser. Select "New" to create a new run configuration for our example. Click on the "Main" tab and verify that the project and class are correct: "examples" and "examples.Example" respectively. Then click on the "Arguments" tab and in the "Program Arguments" window, specify the four command line arguments, all on the same line and in this order:

1. CCRC server URL
2. CCRC server login. If server is a Windows machine, include login domain, e.g.: `domain\login`
3. CCRC server password
4. Path of a non-version-controlled file in a ClearCase web view on the local client machine

Now click the "Run" button at the bottom of the dialog. When executed, the example application should generate output similar to this:

```
##### Processing program arguments
##### Instantiating ClearCase TeamAPI provider
##### Constructing proxy for test file
##### Constructing proxy for test file's parent directory
##### Adding test file to source control: C:\webviews\myview\test.source.vob\testDir\foo.txt
##### Verifying that file is NOT already source controlled...
##### Logging in to CCRC server: http://my-ccrc-server/ccrc
##### Login:    my-domain\my-login
##### Password: ********
##### Verifying that the parent folder IS source controlled...
##### Checking out parent folder...
##### Adding file to source control...
##### Checking in parent folder...
##### Printing test file's properties
##### Resource is controlled:  true
##### Resource is checked out: false
##### Resource's checked in version: \main\1
##### Checked in version created on: Mon Jan 02 11:33:51 EST 2006
##### Done
```