How-To:

Developing with the Pentaho SDK

July 14, 2006

Contents

Introduction

The Pentaho BI Platform SDK is a package consisting of:

- the Pentaho BI Platform source code,
- a sample solution,
- a standalone Eclipse project example,
- sample data,
- and a JBoss application server configured specifically for running the platform.

The purpose of this package is to give developers a set of files that they can drop into an IDE and start compiling and stepping through the Pentaho source code with relative ease. We already distribute all of the content in this package in various other forms, but in response to requests from our community, we have put together this one stop developer's kit. Our goal is to make it easier for our community to extend the platform to meet their needs, without having to suffer great pain getting started.

This SDK (and the related documentation) does not address the Pentaho Design Suite, nor the Report Designer. We intend to have additional how-to's for those projects shortly.

Our IDE of choice is Eclipse. The remainder of this document gives instructions for setting up the SDK in Eclipse specifically, but the project structure should work suitably in other environments as well. We would certainly welcome any contributions to this document that outline how to develop for Pentaho in another IDE.

Intended Audience

The SDK and this documentation is written for the software developer community. We assume the reader has a fair amount of knowledge of the Eclipse platform and Java development.

We will step through two different configuration setups for the platform: developing for a J2EE application and developing for a "stand-alone" Java application that does not require an application server. If the intent is to deploy the platform as a J2EE application, we assume the reader has some experience with J2EE and application servers. For our purposes here, we are demonstrating using the JBoss Application Server.

While not paramount, it will benefit the reader to also have a working knowledge of Apache Ant.

Other Resources

In addition to this document, several other downloads, documentation and resources are available:

Resource	How to Get it
Pentaho Web Site Product information	http://www.pentaho.org
Technical Whitepaper This document describes the architecture of the Pentaho BI Platform and why it is unique	Available as a PDF download http://sourceforge.net/project/showfiles.php?group_id=140317
Advanced Installation Guide This document provides answers to platform installation questions, how to build the platform from source and deployment information.	Available as a PDF download http://sourceforge.net/project/showfiles.php?group_id=140317

Creating Pentaho Solutions This document provides detailed information on how to build Pentaho solutions.	Available as a PDF download http://sourceforge.net/project/showfiles.php?group_id=140317
Product Roadmap Our development roadmap is publicly accessible	http://www.pentaho.org/jira/roadmap.html
Developer Zone This web site provides up-to-date information, discussion forums, F.A.Q.s (some not so 'F' A.Q.'s), and additional design documents.	http://www.pentaho.org Click on 'Developer Zone'

Requirements and Recommendations

Below is a list of the tools you will need to follow this how-to. We recommend that you download all of the necessary packages at the beginning of this exercise. You can then follow the setup instructions uninterrupted.

- The Eclipse platform IDE. The team here is currently using various point versions from the 3.1.X distribution. You can download the Eclipse IDE at http://www.eclipse.org/downloads/index.php.
- A Java SDK. The Pentaho BI Platform is built and tested against Java SDK 1.4, available for many platforms at http://java.sun.com/j2se/corejava/index.jsp. NOTE that you need the full SDK, not just the runtime (JRE).
- The Pentaho SDK. The package you should use with this document is the pentaho-sdk-{latest-date-time}.zip, and can be found at http://www.pentaho.org/download/latest.html.
- If you intend to follow the "Debugging in the JBoss Application Server" exercise, you will want to make sure you do NOT have another application using port 8080 on your computer. This is the default port that the SDK will try to use for the JBoss application server. While you can change the port that the SDK looks to use, you do not want to unnecessarily complicate your life at this point. If port 8080 is in use, we recommend temporarily shutting down the application that is using port 8080 while you are running the SDK's application server.

Getting Started With Eclipse

Let's get started then.

You will need to have Eclipse installed in order to follow along here. If you have not yet downloaded Eclipse, please download it from the URL given above. If you already have Eclipse installed and want to use your current installation and workspace, then you can skip this step and move on to the next section, Getting Started with the JBoss IDE.

Once you have Eclipse downloaded, simply extract the content from the package to your preferred location on your computer. You should see a directory structure similar to the following:

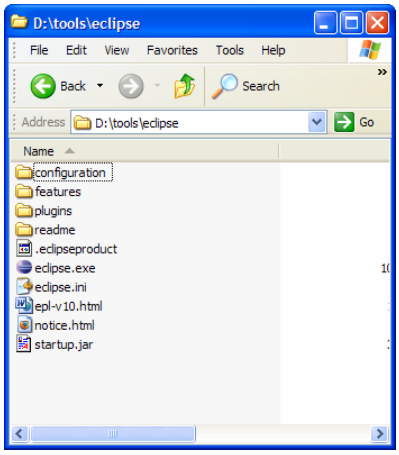

Figure 1. Eclipse project structure

To start Eclipse, execute the eclipse.exe program (or the similar executable on your *nix platform).

You will be prompted for a directory to use as your workspace. The Eclipse workspace is a folder defined on your computer where all of your Eclipse project files, source code and preferences will be stored. Follow the dialogs in Eclipse to specify the folder you would like to use as your workspace.

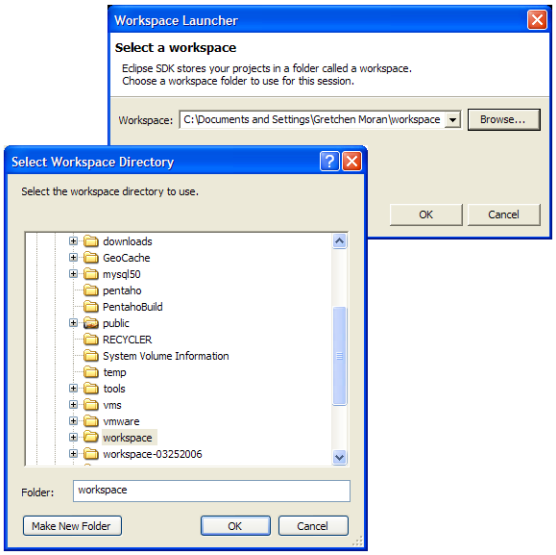

Figure 2. Selecting an Eclipse workspace folder.

Getting Started With the JBoss IDE

If you intend to follow along with the “Debugging in the JBoss Application Server” exercise, then you will want to install the JBoss IDE plug-in for Eclipse. The JBoss IDE plug-in for Eclipse is a tool that allows you to step through Java classes that are running as part of a web application in a JBoss application server.

The instructions for installing the JBoss IDE plug-in can be found at http://docs.jboss.com/jbosside/install/build/en/html/installation.html. This plug-in is retrieved via the Eclipse Update Manager, as described in the install instructions. The Eclipse Update Manager allows you to install and update Eclipse plug-ins directly from the Eclipse platform.

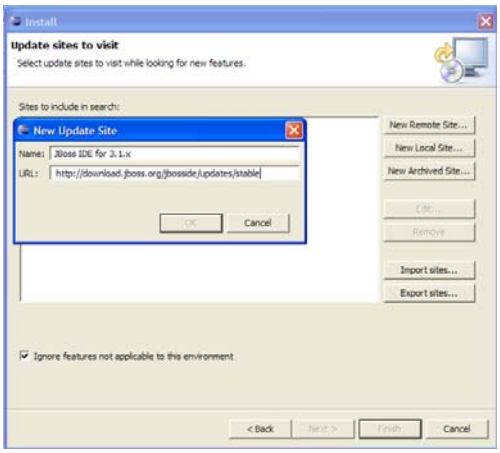

Figure 3. Eclipse Update Manager

Once you have successfully installed the JBoss IDE plug-in, continue to the next section. We will cover how to configure the plug-in for debugging in the "Debugging in the JBoss Application Server" exercise.

The Pentaho BI Platform SDK

The SDK package contains all of the files and project settings to get you up and running in Eclipse. The files are packaged so that you can unpack them directly into your Eclipse workspace.

Using your favorite archive tool (i.e., 7-Zip, WinZip), unpack the Pentaho SDK into your Eclipse workspace folder now.

Here is a snapshot of what you should see in your workspace folder, once you have unpacked the SDK:

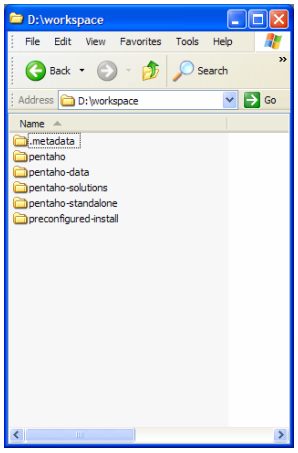

Figure 4. Pentaho SDK files in Eclipse workspace folder.

The *pentaho* Folder

The *pentaho* folder holds the source code, resources and project setting for the Pentaho BI platform. This is the project that you will browse and step through to learn about the platform code and architecture.

The *pentaho-standalone* Folder

The *pentaho-standalone* folder has a simple solution, the platform library and dependencies, and the code for a Java application that runs the platform on its own, without a J2EE application server.

The *pentaho-data* Folder

The *pentaho-data* folder has the HSQLDB files that make up the default databases for our demo solutions, as well as scripts for starting up and shutting down the databases.

The *pentaho-solutions* Folder

The *pentaho-solutions* folder contains all of the xml files, xaction files and other resources that make up the Pentaho demo solutions.

The *pentaho-preconfiguredinstall* Folder

The *preconfigured-install* folder holds a complete JBoss application server, tuned and configured for running the Pentaho BI platform code.

As you can see from the descriptions above, the *pentaho* folder contains the source code for the Pentaho BI Platform. The other three folders (excluding the pentaho-standalone folder) are support folders necessary for running the platform. The idea here is to get you, the developer, looking at how we built our demo applications, which should give you enough knowledge to then either tweak these files for your own solutions or use the platform SDK in your own framework (i.e., Struts, JFaces, HTML, Java application, etc.).

Setting Up the SDK In Eclipse

Now that you have the source code on your computer, you need to tell Eclipse that there are projects defined that you want to see in your Eclipse perspectives.

The Pentaho platform is set up as five separate Eclipse projects. You don't HAVE to set up all five folders as Eclipse projects, but it does help to be able to see all of the files in your Eclipse views and perspectives. We will walk through setting up all of the projects.

Importing the Pentaho Projects

The pentaho, pentaho-data, pentaho-standalone, pentaho-solutions and pentaho-preconfiguredinstall folders all have .project files in them, which are the files that Eclipse uses to configure these folders as projects. These folders can be imported as projects into Eclipse.

1. Start Eclipse and open the Java Perspective.
2. From the File menu, choose the 'Import...' option.
3. From the Import dialog, choose the 'Existing Projects Into Workspace' option.
4. You will be prompted with the Import Projects dialog, which asks you from what directory do you want to import the files. Choose the 'Select Root Directory' option and specify your workspace folder as the root.

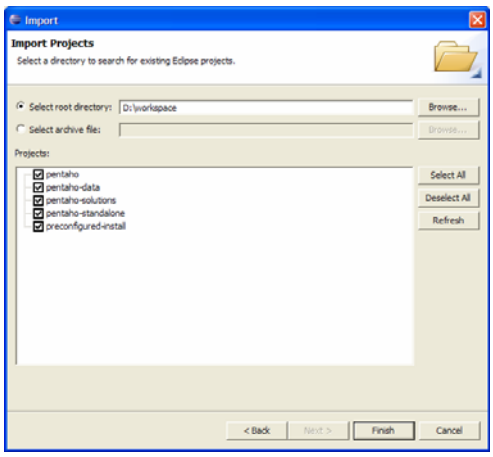

Figure 5. The Import Project dialog in Eclipse.

5. You will see that the Pentaho folders have been selected for import. Choose the Finish button at the bottom of the dialog.

You should now see all of the Pentaho projects in Eclipse.

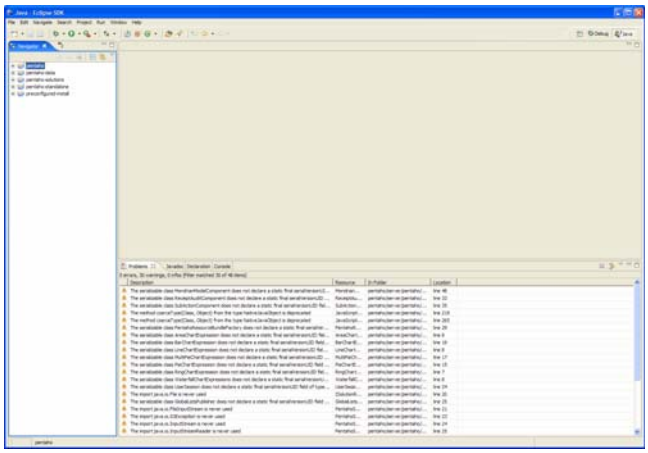

Figure 6. The Pentaho projects in Eclipse.

Eclipse has several different types of projects. Pentaho utilizes two different kinds of Eclipse projects, the simple project and the Java project.

The pentaho-data, pentaho-solutions and pentaho-preconfiguredinstall projects are all Eclipse simple projects. This means that those projects have the most basic Eclipse project configuration and capabilities, which encompasses little more than file browsing.

The pentaho project is an Eclipse Java project. This project is different because, as we mentioned earlier, this is the only project that contains source code that needs compiling. The pentaho project has configuration options and features related to Java development as a result of defining it as a Java project.

Immediately you will notice (if your "Build Automatically" setting is active) that the *pentaho* project is being compiled on import. The project should compile with several warnings, but no compilation errors. Be patient, this task may take up to a minute, depending on the speed of your computer. The *pentaho-standalone* project is also a Java project. It will NOT compile automatically, because there are some setup steps that must be taken before this project can be run. See the next section for details.

If the project does not compile automatically on import, compile it now manually from the Project menu. Select the pentaho project from the view on the left, then from Project menu, choose the 'Build Project' option.

If you get compile errors at this point, we can help! Submit the compilation error that Eclipse is reporting to our forums at http://www.pentaho.org/discussion. Many community members and Pentaho developers are ready to help!

Learning Tool: Debugging in the Standalone Platform Project

The Standalone project, *pentaho-standalone*, is included in the Pentaho BI Platform SDK to provide an example application that harnesses the power of the platform WITHOUT a J2EE application server. The project contains a Java application that runs two action sequences, writing the results of each to a file.

In this Learning Tool, we will walk through the following steps:

1. Setting up the standalone project;
2. Explain the code and resources that make the standalone configuration work;
3. And finally, place a breakpoint in the sample code, to demonstrate debugging through the standalone application.

Standalone Project Setup

The Standalone project is shipped to you with a dependency on the pentaho project. To get started with the Standalone project, you must first run an Ant target, **sample-setup**, that will populate the project with the appropriate libraries for our examples.

To setup the Standalone Project:

1. From the Java Perspective in Eclipse, select the **build.xml** file, located in the root of the *pentaho-standalone* project. I prefer the Navigator view for this (not visible by default), but there are several views that make file selection easy – use the one you like the most.
2. Right-click the build.xml file, and choose the 'Run As...' option, then the 'Ant Build...' option.
3. You will be prompted with an Ant Build dialog. De-select any pre-selected targets, and select ONLY the **sample-setup** target.

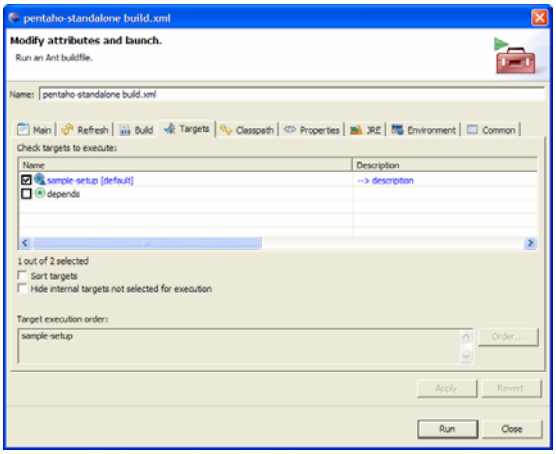

Figure 7. The Ant Build dialog.

4. Choose the 'Run' button at the bottom of the dialog. You should see the activity log from the script in the Console view in Eclipse. Once the target has completed, you will see the message "Build SUCCESSFUL".
5. Select the root folder of the project, pentaho-standalone, in the Eclipse Navigator view. Right-click on the folder, and choose the 'Refresh' option. This will refresh the subfolders so that you can see the files added as a result of building the project.
6. Next let's remove the dependency on the pentaho project. Right-click on the pentaho-standalone project and click Properties. The Properties for pentaho-standalone dialog loads.
7. On the left side of the dialog, select Java Build Path, select the Projects tab, and click the remove button. Notice that pentaho was removed from the list.
8. Next click the Libraries tab and click the Add Jars button. The Jar Selection dialog loads.
9. Select pentaho-standalone and be sure to add every jar in the lib directory.

The project should now be ready to step through, but let's first explain the files that make up the project.

Code and Resources for the Standalone Project

You should see a file structure for the *pentaho-standalone* project similar to the one depicted below. We have built the standalone project with the very least amount of resources (files and source code) necessary, so you can get a feel for what the bare minimum requirements are for running the platform standalone.

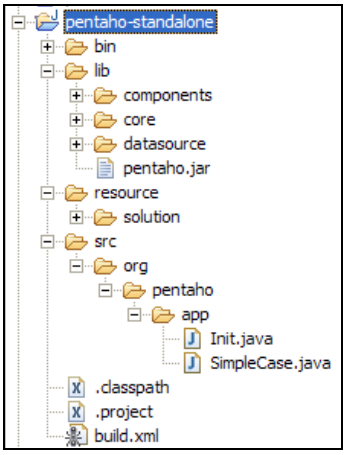

Figure 8. The stand-alone project file structure

The lib Directory

The **lib** directory holds all of the dependency libraries that are needed to run the pentaho-standalone Java application.

The libraries have been separated into the subdirectories to help clarify why each library is included. The libraries under the **components** directory are there specifically for component functionality. The **core** directory holds those libraries needed for the platform to run. The **datasource** directory holds the HSQLDB driver library, and of course the pentaho-<version>.jar is the platform itself.

The resource/solution Directory

The resource/solution directory holds the solution files for our example. These files are very similar to those that are included with our demo. There are two action sequences in our solution, 'Hello World' and 'Simple Report'. If you browse the resource/solution directory, you will see the files for the action sequences.

The Source Files

The **src** directory holds the source code for executing the platform as a standalone Java application.

The org.pentaho.app.SimpleCase.java class is the main class that runs the platform, and ultimately our solution. This class initializes the platform, runs a very simple 'Hello World' action sequence, then runs a very simple JFreeReport action sequence. Both action sequences produce results that are written out to files for the sake of simplicity.

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

    Init.initialize();
    SimpleCase sCase = new SimpleCase();
    sCase.simpleCase( args );

  } catch (Exception e) {
    e.printStackTrace();
  }
}


…


public void simpleHelloWorldCase( String outputPath ) {
  try {
    File f = new File( outputPath + File.separator + "hello_world.txt" );
    FileOutputStream outputStream = new FileOutputStream(f);
    HashMap parameters = new HashMap();
    ISolutionEngine solutionEngine = SolutionHelper.execute( "Simple Case Example", "Hello World",
                                        "getting-started/HelloWorld.xaction", parameters, outputStream  );
  } catch (Exception e) {
    e.printStackTrace();
  }
```

The org.pentaho.app.Init.java class has only one interesting method – the method that initializes the platform:

```
public static boolean initialize() {

  try {

// We need to be able to locate the solution files. in this example, we are using the relative path in our
project/package.
    File solutionRoot = new File( "resource/solution" );

// Create a standalone application context - our application toolbox if you will - passing the path to the solution
files.
    IApplicationContext context = new StandaloneApplicationContext(solutionRoot.getAbsolutePath(),
                                                                   new File(".").getAbsolutePath() );


// Initialize the Pentaho system
    PentahoSystem.init( context );
    return true;

  } catch (Throwable t) {

    t.printStackTrace();
    return false;

  }
}
```

Stepping Through the Sample Code

Stepping through the code in Eclipse should be familiar to you, but we will provide an example here.

Before you attempt to run the SimpleCase class in Eclipse, you must first start the demo databases.

To start the databases:

1. Navigate to the *pentaho-data* folder, either via command line or using your favorite file explorer tool – outside of Eclipse.
2. Under the pentaho-data/demo-data folder there are startup and shutdown scripts for both Windows OS and *nix platforms. Execute the start-up script that is appropriate for your computer (.bat files are for Windows, .sh files are for *nix).

Now let's set a breakpoint and execute the SimpleCase main() method , so that you can see the development process in action.

1. Start by switching to the Eclipse Debug Perspective, if you are not already there.
2. From one of Eclipse's many file exploring views (I use Navigator), navigate in the *pentaho-standalone* project to src/org/pentaho/app/SimpleCase.java, and open that file.
3. Place a breakpoint (from the right-click menu) on line 73 in the file SimpleCase.java. (Note: as code changes this line number could and most likely will change. The line of code that we are breaking on is the line that contains the Init.initialize() code).
4. Right-click on the SimpleCase.java file in your file exploring view. Choose the 'Debug As...' option then the 'Java Application' option. This will launch the class as a Java application.

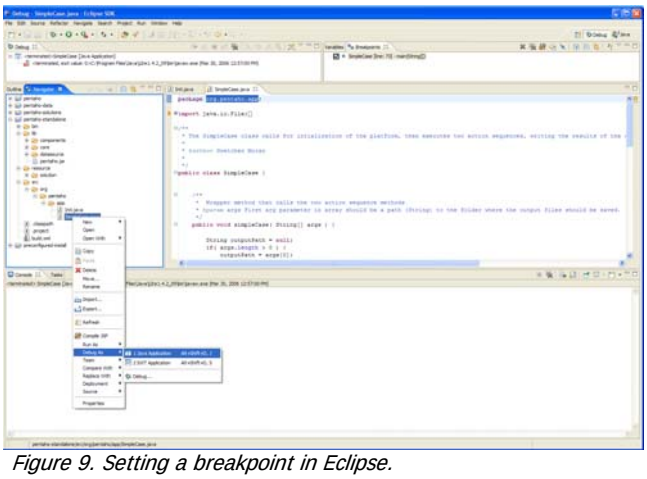

Figure 9. Setting a breakpoint in Eclipse.

5. The program will stop execution at your breakpoint. From here, you can step into, step over or continue execution in Eclipse. Watch the Console view for additional debug messages.

Learning Tool: Debugging in the JBoss Application Server

You are now ready to deploy and debug the platform using the pentaho-preconfiguredinstall project, a JBoss application server.

This Learning Tool consists of the following exercises:

1. We will first walk through the tools that are available to you for deploying the platform Java code and web application files from the *pentaho* project into the pentaho-preconfiguredinstall project.

2. The next step we will take is configuring a server in the JBoss IDE. We will use this to run the JBoss application server with the platform deployed as a web application.
3. Last, we will place a breakpoint in the platform Java code, run the 'Hello World' sample solution, and step through the code from that breakpoint.

Deploying and Updating the Platform Web Application

The pentaho project and the pentaho-preconfiguredinstall project are intended to work together for development purposes. These projects are meant to provide an easy way to take code that is written as part of the platform (in the *pentaho* project) and place it in the right structure as part of a deployed web application in a JBoss server (the *pentaho-reconfiguredinstall* project).

There are two Ant tasks defined in the *pentaho* project's build.xml file that accommodates this development process: they are the **dev-setup** and **dev-update** targets.

The dev-setup Target

The **dev-setup** target is an Ant script that performs the following services:

- Compiles the platform source code out of the *src* folder.
- Buillds a JBoss 4.0.3-compliant war file. If you are not familiar with war files, refer to Sun's documentation on J2EE and web application archives ().
- Copies the HSQLDB library into the pentaho-data project (more about this later).
- Deploys the platform war file into the pentaho-preconfiguredinstall project.
- Deploys the pentaho-style war file into the pentaho-preconfiguredinstall project. This war file contains all of the style resources for the demo web application.
- Deploys the pentaho-portal-layout war file into the pentaho-preconfiguredinstall project. This war file contains all of the portal layout resources for the portal portion of the demo web application.
- Deploys the data definition xml files necessary for the JBoss application server to recognize the HSQLDB databases in the pentaho-data project.

The **dev-setup** target should be run the very first time you set up the Pentaho projects. It populates the preconfigured-install with the platform web application in its entirety.

To run the **dev-setup** target:

1. From the Java Perspective in Eclipse, select the **build.xml** file, located in the root of the *pentaho* project. I prefer the Navigator view for this (not visible by default), but there are several views that make file selection easy – use the one you like the most.
2. Right-click the build.xml file, and choose the 'Run As...' option, then the 'Ant Build...' option. BE SURE you select the option with the ellipses on the end! The 'Ant Build' option builds the **DEFAULT** target, which may not be the **dev-setup** target!
3. You will be prompted with an Ant Build dialog. De-select any pre-selected targets, and select ONLY the **dev-setup** target.

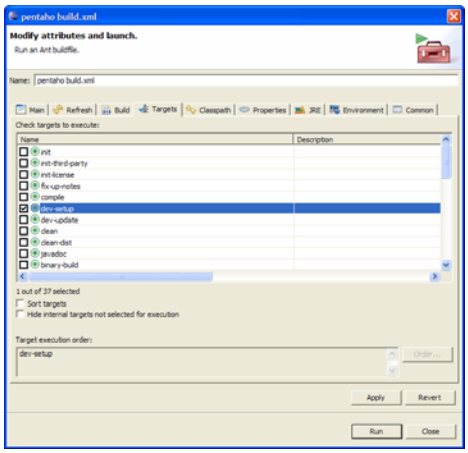

Figure 10. The Ant Build dialog.

4. Choose the 'Run' button at the bottom of the dialog. You should see the activity log from the script in the Console view in Eclipse. Once the target has completed, you will see the message "Build SUCCESSFUL". The *pentaho-preconfiguredinstall* project has now been populated with the Pentaho platform web application.

The dev-update Target

The **dev-update** target is the script that will update the platform web application in the *pentaho-preconfiguredinstall* project with any files that you have made changes to. So, for example, if you add or modify any of the .java files in the *pentaho* project, the *dev-update* target will perform the following services for you:

- Copies any new or modified java classes to the appropriate directory in the *pentaho-preconfiguredinstall* project.
- Copies any new or modified jsps to the appropriate directory in the *pentaho-preconfiguredinstall* project.
- Copies any new or modified xml datasource definitions to the appropriate directory in the *pentaho-preconfigured-install* project.
- Touches the web.xml file in the pentaho web application, which causes the JBoss server to automatically reload the web application and immediately pick up the changes.

This target should be run when you have made changes to the source code and you want to test those changes in the JBoss application server.

You can run the **dev-update** target by following the instructions outlined above for the **dev-setup** target, but choosing the **dev-update** target of course.

Configuring the JBoss IDE Server

Now that you have the pentaho web application copied into the *pentaho-preconfiguredinstall* project, it's time to start up the JBoss application server and make sure everything works.

The JBoss IDE is a pretty sweet tool. It's easy to configure, and once that's done, all you have to do is start the server through the IDE and you can debug Java code.

To configure a new server in the JBoss IDE:

1. Switch to the Eclipse Debug Perspective.

2. From the Window menu, choose the ‘Show View’ option, then the ‘Other…’ option.
3. You will be prompted with a dialog populated with named views. Choose the ‘JBoss IDE’ option, then the ‘Server Navigator’ view.

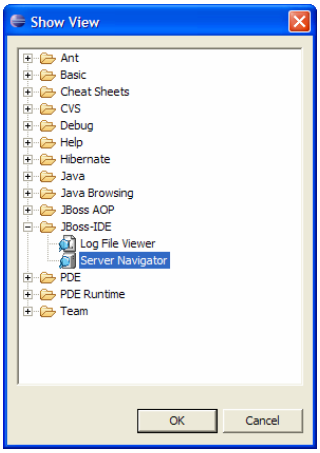

Figure 11. Selecting the JBoss IDE Server Navigator view.

4. Choose the OK button to finish. You should see the Server Navigator view appear in the bottom of the Eclipse Debug Perspective.

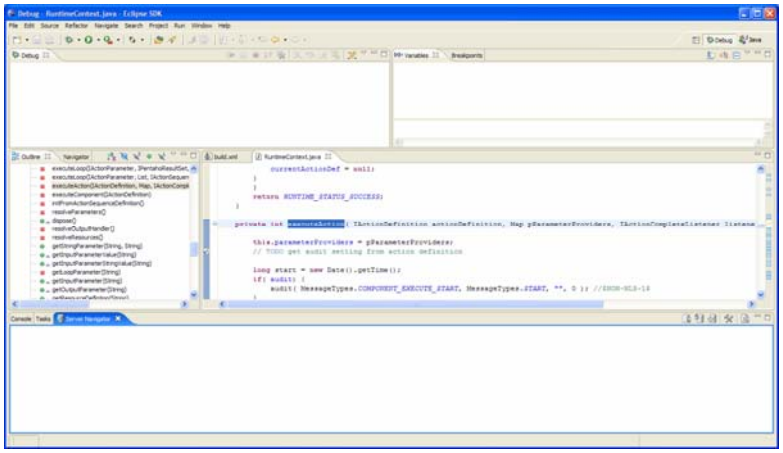

Figure 12. The Server Navigator view in the Eclipse Debug Perspective.

5. Right-click in the Server Navigator window pane. Choose the ‘Configuration…’ option.
6. You will be prompted with a Configuration dialog. In the left pane, choose the ‘JBoss 4.0.x’ option, then choose the New button under the left pane. You should see a dialog box similar to the one shown here.

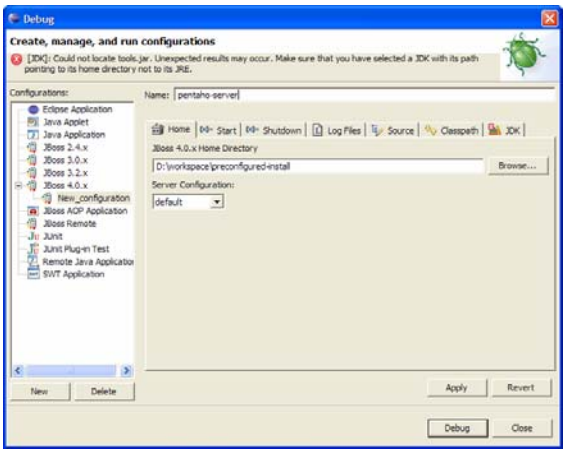

Figure 13. JBoss Server Configuration dialog.

7. Enter ‘pentaho-server’ as the name of your new configuration.
8. Choose the ‘Browse…’ button and navigate to the pentaho-reconfiguredinstall folder. Set this folder as the JBoss 4.0.x Home Directory.
9. Select ‘default’ as the Server Configuration.
10. Choose the ‘Apply’ button to save this configuration.
11. Switch to the ‘Source’ tab in the dialog.
12. Choose the ‘Add’ button, then ‘Java Project’ from the next dialog, then ‘pentaho’ from the last dialog. Choose OK.

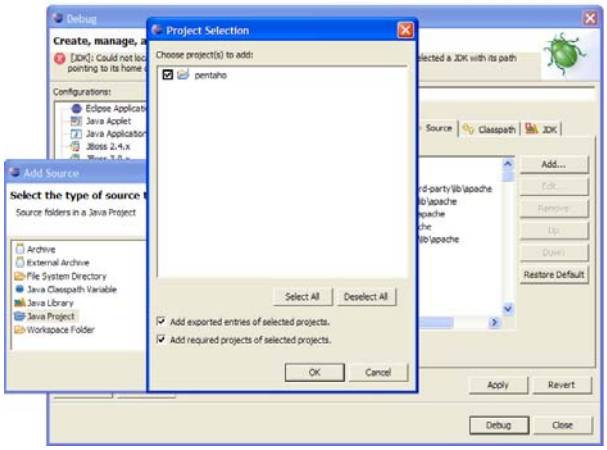

Figure 14. Connecting source code to the JBoss Server configuration.

13. Choose the ‘Apply’ button to save this configuration.
14. Select the ‘Close’ button to close the dialog.

You should now see the ‘pentaho-server’ listed in your Server Navigator view. You can start and stop the pentaho-server by right-clicking on the pentaho-server entry in the Server Navigator view.

NOTE!!! Before you start the *pentaho-server*, you must first start the databases for the demo to run properly. Proceed to the next section for instructions on starting the databases.

Stepping Through ‘Hello World’

All that’s left to do is start up the platform, set our breakpoint and watch it all work!

As noted in the last step, before you attempt to start the *pentaho-server* in Eclipse, you must first start the demo databases.

To start the databases:

1. Navigate to the *pentaho-data* folder, either via command line or using your favorite file explorer tool – outside of Eclipse.
2. Under the pentaho-data/demo-data folder there are startup and shutdown scripts for both Windows OS and *nix platforms. Execute the start-up script that is appropriate for your computer (.bat files are for Windows, .sh files are for *nix).

Start the pentaho-server now:

1. From the Eclipse Debug Perspective, Server Navigator view, right-click on the pentaho-server entry.
2. Choose the 'Start' option.

You will see many start-up messages and warnings scroll by in the Console view. As long as the last message on startup is "`Pentaho BI Platform server is ready.`", then you know the platform is up and running successfully.

Now let's set a breakpoint and execute our 'Hello World' demo solution, so that you can see the development process in action.

1. Start by switching to the Eclipse Debug Perspective, if you are not already there.
2. From one of Eclipse's many file exploring views (I use Navigator), navigate in the *pentaho* project to server/pentaho/src/org/pentaho/core/runtime/RuntimeContext.java, and open that file.
3. Place a breakpoint (from the right-click menu) on line 878 in the file RuntimeContext.java. (Note: as code changes this line number could and most likely will change. The line of code that we are breaking on is the line that contains the executeComponent() code).

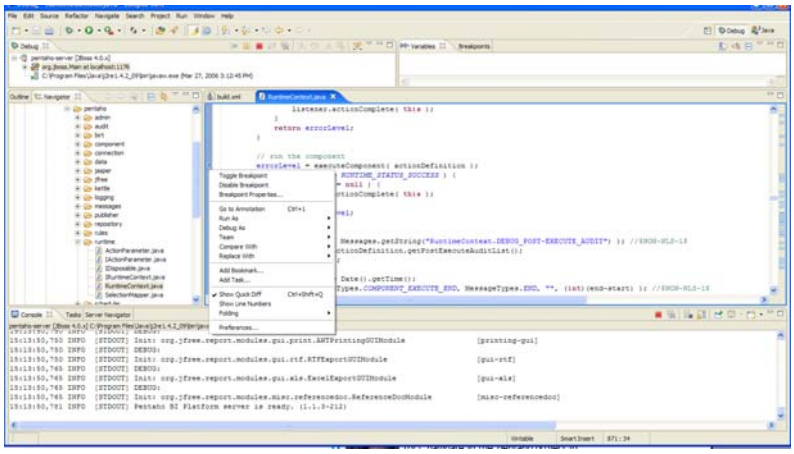

Figure 15. Setting a breakpoint in Eclipse.

4. Next, open a web browser window. Navigate to http://localhost:8080/pentaho/ViewAction?&solution=samples&path=getting-started&action=HelloWorld.xaction.

5. Your breakpoint in Eclipse should be hit before the result of the component execution is sent from the server back to the browser. From here, you can step into, step over or continue execution in Eclipse. Watch the Console view for additional debug messages.