Space Details

Key:	PentahoDoc
Name:	BI Server Documentation - Latest
Description:	Latest version of the Pentaho BI Server
Creator (Creation Date):	admin (Nov 15, 2006)
Last Modifier (Mod. Date):	admin (Nov 27, 2006)

Available Pages

- Building and Debugging Pentaho with Eclipse
  - 01. Setting Up Your Environment
  - 02. Exploring the Pentaho Repository
  - 03. Creating Repository Connections
  - 04. Checking Out Projects
  - 05. Exploring the build.xml
  - 06. Debugging with JUnit
  - 07. Debugging with the Standalone Platform Project
  - 08. Debugging with the JBoss IDE

Building and Debugging Pentaho with Eclipse

This page last changed on Nov 30, 2006 by bhagan.

The purpose of this document is to instruct developers on how to begin developing with Pentaho. This document explains how to set up a development environment in Eclipse, including how to make a connection to the Pentaho Subversion repository, checkout the Pentaho projects, build them, and debug them using a "stand-alone" Java application and using the Eclipse JBoss IDE. The document also explains the Pentaho repository structure and a number of build targets important to developing with Pentaho.

The intended audience is the software developer community. We assume the reader has a fair amount of knowledge of the Eclipse platform and Java development. 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.

If after painstakingly following the instructions in this guide you still do not reach your final destination, reach out! Help is only a forum away at http://forums.pentaho.org/.

01. Setting Up Your Environment

This page last changed on Apr 03, 2007 by dkincade.

02. Exploring the Pentaho Repository

Below is a list of the tools you will need to get started with Pentaho. We recommend that you download all of the necessary packages now so you can follow the trail without interruption.

- A Java SDK. The Pentaho BI Platform is built and tested against Java SDK 1.4. NOTE that you need the full SDK, not just the runtime (JRE).
- Ant, the Java-based build tool.
- The Eclipse platform IDE.
- Subclipse an Eclipse plugin that adds Subversion integration to the Eclipse IDE.
- The JBoss IDE for Eclipse.

To get setup:

1. Download and install your Java SDK.
2. Downlaod and install Ant.
3. Download and install Eclipse.
4. Download and install Subclipse.
5. Download and install the JBoss IDE for Eclipse.

The sections that follow are notes about which versions of each of these tools that the Pentaho team uses, and tips on getting the environment situated for developing in the Pentaho platform codeline. Each tool has detailed setup instructions included with their respective download, so we won't re-write every step here.

Java SDK

The Pentaho BI Platform is built and tested against Java SDK 1.4. NOTE that you need the full SDK, not just the runtime (JRE). You will see in later steps that you must specify the Java compiler executable (javac) for the Ant build scripts to function properly. This executable lives in the /bin directory of the Java SDK that you install, so make a note to remember where you have the SDK installed.

Ant

Ant is a Java-based build tool that we use to build the platform, as well as support development activities for Pentaho within Eclipse. The version currently used to build the downloadable binaries for Pentaho is Ant 1.6.5. We recommend that you have the Ant tool available to build some of the Pentaho Ant targets from the command line (necessary in some certain circumstances).

A version of Ant gets distributed with the Eclipse IDE. This version is used by the IDE for build features, and may be modified for Eclipse plugin purposes. To determine the version that is distributed with the Eclipse IDE, navigate from the Window menu at the top of Eclipse to Preferences... | Ant | Runtime, and checkout the versioning in the plugin's path.

Eclipse IDE

Only recently has the version of Eclipse been an issue when setting up your Pentaho development environment, and it has nothing to do with the Pentaho code. If you want to run and debug the code in the JBoss application server, you will need the JBoss IDE (see below), and therein lies some compatibility issues (the latest version of the IDE seems to have some troubles with the latest version of Eclipse). The short answer is that the following versions/distributions are being used by the Pentaho team, so we know these configurations work.

1. Download Eclipse pre-bundled with the latest JBoss IDE plugin from the JBoss site - Eclipse IDE version 3.2.1 bundled with JBoss IDE version 2.0.0beta2. **OR**
2. Download Eclipse 3.2.1 M20060921-0945 from the Eclipse site; then download and install the JBoss IDE plugin version 2.0.0 beta2 from JBoss.

Subclipse

Subclipse is a plugin to Eclipse that enables Subversion features within the Eclipse environment. Subversion is the source control repository used by most Pentaho projects. Subclipse takes advantage of the Eclipse Update Manager that allows you to register with a URL and retrieve and install plugins automatically within the Eclipse IDE. The version of Subclipse that is compatible with our configuration is Subclipse 1.2.x. You can find detailed install instructions on the Tigris.org site, or follow the abbreviated instructions below:.

In Eclipse, select from the menu bar Help | Software Updates | Find and Install...

1. In the first step of the wizard, select "Search for new features to install". Click the "next" button.
2. In the next step, click the button "New Remote Site...".
3. In the popup, give the site the name of your choosing, and enter the following url: http://subclipse.tigris.org/update_1.2.x;. Click the "OK" button.
4. You should see a list of features to choose from in the next step. Choose the "Subclipse plugin" entry that is appropriate for your version of Eclipse.
5. Allow the plug-in to finish installing. For detailed instructions on using Subclipse, go to the Eclipse Help menu, choose Help Contents, then browse the Subclipse tree in the Table of Contents.

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. If you downloaded the bundled Eclipse\JBoss IDE package, then you do not need to install the JBoss IDE plugin - you already have it. 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 JBoss IDE plug-in is typically retrieved via the Eclipse Update Manager. However, at the time of this writing the JBoss IDE version 2.0.0beta2 doesn't have install instructions, documentation or a hook into the Update Manager. That's what we get for using a development build  This shouldn't be a problem, it is pretty straight forward, and we'll do our best to describe it here.

Get the JBoss plug-in here:

1. Download the plugin distribution, JBossIDE-2.0.0.Beta2-ALL.zip (not the plugin\Eclipse bundle!).
2. Unzip the bundle into your eclipse installed directory - not that the directory structure in the zip file starts at /eclipse .
3. Start\restart Eclipse.

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.

02. Exploring the Pentaho Repository

Quick Links For This Document

02. Exploring the Pentaho Repository

This page last changed on Nov 30, 2006 by bhagan.

01. Setting Up Your Environment | 03. Creating Repository Connections

The Pentaho BI Platform code is housed at svn://source.pentaho.org/ in a Subversion repository. If you have no experience with Subversion, have no fear - there's an online book at http://svnbook.red-bean.com.

The Pentaho projects in Subversion are:

- pentaho - the main Pentaho source code tree
- pentaho-cubedesigner - source for the wizard-driven, graphical user interface used for connecting to relational data sources, defining analytical dimensions, and identifying measures or "facts" for analysis.
- pentaho-data - the default sample databases
- pentaho-designstudio - source for the graphical eclipse environment for building and testing Action Sequence documents
- pentaho-designstudioIDE - essentially eclipse with a build file that will install the pentaho action-sequence-plugin and splash screen
- pentaho-preconfiguredinstall - source for the preconfigured Jboss application server
- pentaho-reportdesigner - source for the standalone designer for creating JFreeReports
- pentaho-reportwizard - source for the standalone wizard for creating JFreeReports
- pentaho-solutions - the sample solutions
- pentaho-standalone - the code for a Java application that runs the platform on its own, without a J2EE application server

We've tried diligently to keep our project structures simple for ourselves and the sanity of our community. Below you will find descriptions of the projects that we will be discussing throughout this document:

- pentaho
- pentaho-data
- pentaho-solutions
- pentaho-preconfiguredinstall
- pentaho-standalone

We will not be discussing the Pentaho client tools in this document:

- pentaho-cubedesigner
- pentaho-designstudio
- pentaho-designstudioIDE
- pentaho-reportdesigner
- pentaho-reportwizard

The pentaho Project

The *pentaho* project 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.

Directory/File	Description
/pentaho	Root directory of source tree.
/cobertura	Contains libraries used to determine what percentage of code is covered by unit tests.
/scripts	Contains startup scripts used when building the Preconfigured Install
/server	Source tree for server code.
/third-party	Contains the lib directory; holds all third party libraries necessary for compilation and building.
.classpath	Eclipse file for setting the project classpath.
.project	Eclipse file describing the project.
build.xml	Ant build file for Pentaho project.
default.properties	Properties for building the Pentaho project; override properties in this file by creating an override.properties file in the same directory, and adding those properties you want to override.
deployment_build.properties	Properties file used by the deployment_build.xml
deployment_build.xml	Build file that contains targets used to build appserver specific deliverables. It is used by build.xml and is delivered as part of the j2ee deployments.
excludejars.generic	Text file listing those jars that should not be included in the .war file for the tomcat-war target in the build file.
excludejars.jboss	Text file listing those jars that should not be included in the .war file for the jboss-war target in the build file.

The pentaho-data Project

The Pentaho BI Platform ships with a set of default databases (using Hypersonic SQL) to hold the necessary repositories for several feature components and subsystems. In order to get these parts of the platform up and running you will need these repositories. This data exists under the module 'pentaho-data'. Pentaho-data also includes startup and shutdown scripts for the databases.

Directory/File	Description
/pentaho-data	Root directory of database tree.
/hibernate	Holds the platform repository scripts for Hypersonic SQL.
/quartz	Holds the Quartz scheduler scripts for Hypersonic SQL.

/sampledata	Holds the sample data scripts for Hypersonic SQL.
/shark	Holds the Shark workflow scripts for Hypersonic SQL.
start_hypersonic.bat	Startup batch file for Windows platform; used to start the databases on Windows.
start-hypersonic.sh	Startup batch file for *nix platform; used to start the databases on *nix.
stop_hypersonic.bat	Shutdown batch file for Windows platform; used to shutdown the databases on Windows.
stop-hypersonic.sh	Shutdown batch file for *nix platform; used to shutdown the databases on *nix.

Important

The 'pentaho-data' module is structured to be a standalone component that acts as a database server. The scripts provided in the project start and shutdown the Hypersonic databases. The scripts rely on the Hypersonic jdbc classes being available in a /lib directory under the 'pentaho-data' directory. So in order to get these scripts running properly, you will need to place the Hypersonic driver jar where the scripts expect it as follows

1. Create a directory named lib under the 'pentaho-data' directory.
2. Copy the Hypersonic jdbc driver .jar file (hsqldb.jar) into the lib directory. We recommend that you retrieve this file from the third-party/lib directory in the 'pentaho' project, as that one will be sure to be the right version.

The pentaho-preconfiguredinstall Project

The *preconfigured-install* project holds a complete JBoss application server, tuned and configured for running the Pentaho BI platform code. For details on the JBoss application server directory structure, see the JBoss Wiki

The pentaho-solutions Project

If your goal is to set up the platform as a web application, or simply have a starting point for your own solutions, you will want to get the latest samples from Subversion. Setting up the samples is a relatively trivial exercise, and provides a great way to test the various components in your deployment.

The samples are located under the project named "pentaho-solutions". The pentaho-solutions project consists of a set of samples that demonstrate several component features of the platform and a set of clean configuration files.

To run the samples, you will also want to download the default Hypersonic databases. See the previous section for instructions on getting the databases.

Directory/File	Description
/pentaho-solutions	Root directory of solution tree.

/admin	Root directory for the administrative samples.
/samples	Root directory for sample solution.
/system	The system directory contains all platform and component configuration information.
/test	Root directory for test solution.

Important

Download the samples to a directory that is a sibling of your application server's root directory. The "pentaho-solutions" directory will be found by the web application without configuration changes as long as it is deployed as the sibling to the app server as described above.

The pentaho-standalone Project

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.

Directory/File	Description
/resource	The resource 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.
/src	The **src** directory holds the source code for executing the platform as a standalone Java application.
/.classpath	Eclipse file for setting the project classpath.
/.project	Eclipse file describing the project.
/build.xml	Ant build file for Pentaho project.

01. Setting Up Your Environment

03. Creating Repository Connections

Quick Links For This Document

03. Creating Repository Connections

This page last changed on Mar 01, 2007 by gmoran.

02. Exploring the Pentaho Repository | 04. Checking Out Projects

Eclipse is the IDE of choice for the Pentaho team, and we've been using Subclipse as our Subversion plugin. The plugin provides a "SVN Repository Exploring Perspective", from which you can create a connection.

To create a connection using the Subclipse plugin:

1. In the SVN Repository Exploring perspective, right click, point to New, and select Repository Location. The Add New Repository Location dialog loads.
2. In the Url combo box, enter svn://source.pentaho.org/svnroot.
3. Click Finish. You now have a connection to the Subversion repository.

Expanding the location, you will see the projects. As is standard with Subversion repositories, each project has three child directories: branches, tags, and trunk. The project/trunk directory has the latest code, and it is probably what you want to checkout. For a complete explanation of the "pentaho" project structure, see 02. Exploring the Pentaho Repository Structure.

02. Exploring the Pentaho Repository | 04. Checking Out Projects

Quick Links For This Document

04. Checking Out Projects

This page last changed on Feb 26, 2007 by gmoran.

03. Creating Repository Connections 05. Exploring the build.xml

Eclipse has several different types of projects, and Pentaho utilizes the simple project and the Java project. Simple projects have the most basic Eclipse project configuration and capabilities, which encompasses little more than file browsing. Java projects contain source code that needs compiling and configuration files such as the .classpath file, which will set up the build libraries found in the \lib sub-directory of the source tree. The pentaho-data, pentaho-solutions and pentaho-preconfiguredinstall projects are all Eclipse simple projects. The pentaho project and the pentaho-standalone project are both Eclipse Java projects.

The recommended structure for the modules you are pulling from Subversion is to download them as sibling directories - this structure will accommodate the platform in finding your solutions directory automatically, and also make keeping track of the different modules a bit easier.

Immediately after checking out the pentaho project, you will notice (if your "Build Automatically" setting is active) that the project is being compiled. The project may compile with warnings, but not with compilation errors. Be patient, this task may take up to a minute, depending on the speed of your computer. The pentaho-standalone project will NOT compile automatically, because there are some setup steps that must be taken before this project can be run.

To complete the examples in this document, you will need to checkout the following projects:

- pentaho
- pentaho-data
- pentaho-solutions
- pentaho-preconfiguredinstall
- pentaho-standalone

Retrieving Different Codelines

At this point, you may want to retrieve the latest code for the platform, which is the codeline that is being built for our next release, version 1.6. Or, if you are investigating a problem with the latest production release, you will want version 1.2.x. The next two sections explain how to get one the 1.2 branch of the code, versus the latest, which is, by Subversion terminology, "the trunk".

Either way, to retrieve the source, simply issue a command to Subversion either from a command line or via your favorite Subversion client application. We've been using Subclipse.

Get the Latest Code

In the SVN Repository View, expand your repository connection. Earlier, we created svn://source.pentaho.org/svnroot.

1. Expand a project. In this example, we will use pentaho.

2. Right click **trunk**.
3. From the right-click menu, select Checkout... The Checkout from SVN wizard loads.
4. Follow the instructions in the wizard to checkout the project to your workspace. Name the new Eclipse project the same name that was used for the project in Subversion.
5. Repeat these steps for each project listed above.

After you checkout the pentaho project, if it does not compile automatically, compile it manually from the Project menu. Select the pentaho project from the view on the left, then from Project menu, choose the 'Build Project' option. Only the pentaho project needs to be compiled, the rest of the projects are simple projects that do not need to be built.

If you do experience errors, take the necessary steps to resolve them before moving forward. The typical reason for compilation errors when setting the project up for the first time is either the module structure does not follow the recommended hierarchy, or a third party library is missing from the /lib directory or the .classpath file. If you're still stuck, we can help! Submit the compilation error that Eclipse is reporting to our forums at http://forums.pentaho.org. Many community members and Pentaho developers are ready to help!

Get the Version 1.2.x Codeline

In the SVN Repository View, expand your repository connection. Earlier, we created svn://source.pentaho.org/svnroot.

1. Expand a project. In this example, we will use pentaho.
2. Expand the **branches** tree.
3. Right click **1.2**.
4. From the right-click menu, select Checkout... The Checkout from SVN wizard loads.
5. Follow the instructions in the wizard to checkout the project to your workspace. Name the new Eclipse project the same name that was used for the project in Subversion.
6. Repeat these steps for each project listed above.

After you checkout the pentaho project, if it does not compile automatically, compile it manually from the Project menu. Select the pentaho project from the view on the left, then from Project menu, choose the 'Build Project' option. Only the pentaho project needs to be compiled, the rest of the projects are simple projects that do not need to be built.

If you do experience errors, take the necessary steps to resolve them before moving forward. The typical reason for compilation errors when setting the project up for the first time is either the module structure does not follow the recommended hierarchy, or a third party library is missing from the /lib directory or the .classpath file. If you're still stuck, we can help! Submit the compilation error that Eclipse is reporting to our forums at http://forums.pentaho.org. Many community members and Pentaho developers are ready to help!

03. Creating Repository Connections 05. Exploring the build.xml

Quick Links For This Document

05. Exploring the build.xml

This page last changed on Apr 26, 2007 by bhagan.

04. Checking Out Projects

06. Debugging with JUnit

We recently revamped our build process, which included separating the production build from our development build, as they satisfy different requirements. In both the 1.2.x and the trunk codelines, the production build uses the build.xml, and when you're developing inside Eclipse, you can use the dev_build.xml.

The dev_build.xml

The build file that you should use to develop with under the latest codeline is the **dev_build.xml**. This build file is written specifically to accomodate developers that are running and debugging against the preconfigured install JBoss demo server (the pentaho-preconfiguredinstall project contains the shell JBoss server application).

The dev_build.xml is dependent on the dev_build.properties file for parameter information. There are several parameters that you may want to override to accommodate your environment. To set these overrides, do not modify the dev_build.properties file! Instead you will want to create an override.properties file and specify your override parameters there. That way, when changes are made to these files, you don't have to merge your changes to the properties.

Target: dev-setup

The dev-setup target is the target that you should be mainly concerned with. Once you have the source for the four main Pentaho projects, the dev-setup Ant target creates a fully populated, configured and deployed JBoss app server and Pentaho demo application. Here are the steps that the dev-setup performs:

- Compiles all source code
- Creates a pristine copy of the pentaho-preconfiguredinstall project, which is the JBoss app server shell. To customize the location of this copy, set the **target.server.dir** property in your override.properties file.
- Creates a pristine copy of the pentaho-solutions project, which contains the sample solutions for the PCI demo. To customize the location of this copy, set the **target.solutions.dir** property in your override.properties file.
- Copies all compiled classes, .war files, datasource configuration files and miscellaneous jars that JBoss needs to run the Pentaho application server to the target server directory.
- Copies the HSQLDB driver jar tot eh pentaho-data project. The pentaho-data project is set up to act as a stand alone database server, which is why the driver jar needs to be accessible in the specified location, so that the startup and shutdown scripts work properly.
- 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.

The first time the target is run, all of the above actions are executed, setting up a debuggable

environment. On subsequent runs, only modified files are replaced. 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.

Target: clean-target-server

The reason the dev_build targets are architected the way they are, is so that you can be sure that your code changes are picked up in the built debugging environment. You also want to be sure that you are working in an environment that is free of residual files being left around from previous runs or builds. The clean-target-server target deletes the entire target server directory, so that the next time dev-setup is run, you once again have a pristine deployment.

Other Important Property Overrides

In order for the build to run successfully in your environment, there may be a couple of other properties you want to add to your override.properties file:

- **javac.path** - specify the path to your Java compiler (javac.*)
  Example: javac.path=d:/tools/j2sdk1.4.2_07/bin/javac
- **java1.4.home** - if java 1.5 SDK and 1.4 SDK are installed, and java 1.5 is your default, specify java1.4 home, since the platform SDK recommendation is 1.4
  Example: java1.4.home=D:/tools/j2sdk1.4.2_07

04. Checking Out Projects

06. Debugging with JUnit

Quick Links For This Document

- 01. Setting Up Your Environment
- 02. Exploring the Pentaho Repository
- 03. Creating Repository Connections
- 04. Checking Out Projects
- 05. Exploring the build.xml
- 06. Debugging with JUnit
- 07. Debugging with the Standalone Platform Project
- 08. Debugging with the JBoss IDE

06. Debugging with JUnit

This page last changed on Apr 05, 2007 by dkincade.

05. Exploring the build.xml

07. Debugging with the Standalone Platform Project

The easiest method for tracing through code in the Pentaho project is to use a JUnit test case. (If you are not familiar with JUnit, you can learn more about it at http://www.junit.org/index.htm) . There are a number of JUnit test cases already built to test individual component features and subsystems of the platform. They can be found in the pentaho project's source tree at ./server/pentaho/test/org/pentaho. Eclipse has built in capabilities for running JUnit tests. Just select the test case that you wish to run (open the source file), and from the Eclipse menu, select Run..New JUnit Test.. Follow the instructions provided in the Eclipse dialog, which are pretty straight forward.

05. Exploring the build.xml

07. Debugging with the Standalone Platform Project

Quick Links For This Document

07. Debugging with the Standalone Platform Project

This page last changed on Mar 02, 2007 by gmoran.

06. Debugging with JUnit 08. Debugging with the JBoss IDE

The Standalone project, *pentaho-standalone*, is 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 section, 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 contains 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.
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.

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

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 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.
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.

06. Debugging with JUnit 08. Debugging with the JBoss IDE

Quick Links For This Document

08. Debugging with the JBoss IDE

This page last changed on Jun 05, 2007 by bseyler.

07. Debugging with the Standalone Platform Project

If you have followed the previous setup steps, you are now ready to debug the platform. We are assuming that you have the necessary projects and a configured JBoss server to run as we outline here. Here's the steps we are going to perform:

1. We will configure a server in the JBoss IDE. We will use this to run the JBoss application server with the platform deployed as a web application.
2. 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.

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 for the JBoss application server. While you can change the port that the server 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 application server.

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.

Configuring a Server in JBoss IDE 2.x.x

These instructions work for anyone running the recommended environment as described in Setting Up Your Environment doc.

To configure a new server in the JBoss IDE:

First, we need to setup a server runtime in the Eclipse Preferences.

1. Open the Preferences dialog from the Window menu.
2. Find the 'Server | Installed Runtimes' entry in the tree on the left.
3. In the pane on the right, click the 'Add' button.
4. For runtime type, select 'JBoss Server Adapter Runtime' under 'JBoss Inc'. (If there are multiple choices such as JBoss 4.0 Server Adapeter Runtime, select the 4.0 version)
5. Choose next. On the next pane, set the following parameters:
   a. Name your runtime 'PCI Demo'.
   b. This part is IMPORTANT! For the Home directory, **navigate the the target server directory that is specified in your dev_build (or override) properties file**! That is the server you

want to debug against, as that is where the webapp is deployed to. (NOTE: you can make sure you get the correct property value by looking at the **pentaho/dev_build.properties** file and finding the value of the property named **target.server.dir**)
    c. Switch your JRE to the 1.4 JRE. If you don't have a 1.4 JRE configured with Eclipse, do that now.
6. Click OK to close the dialog.

Now we can configure a new server:

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 'Server' option, then the 'JBoss Server View' view.
4. Choose the OK button to finish. You should see the JBoss Server View appear in the bottom of the Eclipse Debug Perspective.
5. Right-click in the JBoss Server View window pane. Choose the 'New...| Server' option.
6. You are now in the Server setup wizard. Set the parameters for the wizard as follows:
    a. Server's host name: localhost
    b. Server type: Under 'JBoss Inc', select JBoss AS 4.0
    c. Server runtime: Select the 'PCI Demo' runtime that we set up earlier.
    d. Choose next, and give your server a unique name, anything you like.
    e. Choose finish.

And last, we will tailor our server configuration:

1. In the JBoss Server View, you should see your new server, and two panes stacked vertically. Right click on the name of your server in the **bottom** pane. Select the menu item 'Edit Launch Configuration'.
2. In the new dialog, switch to the 'Start Args' tab. Add the following arguments to the VM arguments section:
   -Xms512m -Xmx1028m
3. On the Source tab, click the add button and select the pentaho project. This will automagically identify all source in the pentaho project as source for this server config.
4. On the JRE tab, specify the 1.4 JRE, if it is not selected already.

You should now see the name of your new server listed in your JBoss Server View. You can start and stop the pentaho server by right-clicking on the entry in the JBoss Server View. NOTE!!! Before you start the server, you must first start the databases for the demo to run properly. Proceed to hello_world for instructions on starting the databases.

Configuring a Server in JBoss IDE 1.x.x

These instructions are included for anyone who is setup with an older version of the JBoss IDE than what is recommended in the Setting Up Your Environment doc.

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.
4. Choose the OK button to finish. You should see the Server Navigator view appear in the bottom of the Eclipse Debug Perspective.# Right-click in the Server Navigator window pane. Choose the 'Configuration...' option.
5. 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.# Enter 'pentaho-server' as the name of your new configuration.
6. Choose the 'Browse...' button and navigate to the pentaho-reconfiguredinstall folder. Set this folder as the JBoss 4.0.x Home Directory.
7. Select 'default' as the Server Configuration.
8. Choose the 'Apply' button to save this configuration.
9. Switch to the 'Source' tab in the dialog.
10. Choose the 'Add' button, then 'Java Project' from the next dialog, then 'pentaho' from the last dialog. Choose OK.
11. Choose the 'Apply' button to save this configuration.
12. 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! Before you attempt to start the *pentaho-server* in Eclipse, you must first start the demo databases.

To start the databases:

1. Either via command line or using your favorite file explorer tool - outside of Eclipse - navigate to the *pentaho-data* folder.
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 (or the JBoss Server View), right-click on the pentaho-server entry.
2. Choose the '**Debug**' 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/RuntimeContextBase.java, and open that file.
3. Place a breakpoint (from the right-click menu) on line 1092 in the file RuntimeContextBase.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).

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.

07. Debugging with the Standalone Platform Project

Quick Links For This Document