Oracle® Java Embedded Suite

Application Developer's Guide

Release 7.0

E28526-01

November 2012

ORACLE®

Application Developer's Guide, Release 7.0

E28526-01

Contents

List of Figures

List of Tables

Preface

Oracle Java Embedded Suite is a collection of components that assist the development of embedded Java applications incorporating database, web application, or RESTful web service technologies.

Audience

This document is intended for application developers who are working with Oracle Java Embedded Suite.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

Oracle customers have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

Related Documents

For more information, see the following document: *Oracle Java Embedded Suite Release Notes.*

Component chapters in this guide have links to relevant component documentation.

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.

Convention	Meaning
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

1 Overview

This chapter summarizes the components of the Oracle Java Embedded Suite.

Table of Contents

1.1 About the Oracle Java Embedded Suite

The Oracle Java Embedded Suite adds database, RESTful web services, and servlet facilities to the Java SE for Embedded devices Java runtime environment. Chapter 8 describes samples of all services, working together and separately. The code for these samples is available in the companion samples bundle.

For the version numbers of all components, refer to the *Release Notes*.

1.2 About Oracle Java SE Embedded

Oracle Java SE Embedded is a large subset of the desktop standard Java SE. Graphical interfaces, such as Swing and AWT are not provided so that memory usage is consistent with the needs of mid-size headless embedded devices. In addition, Oracle Java SE Embedded benefits from many space optimizations, while maintaining API compatibility with Java SE. The secure execution environment, automatic garbage collection, and other familiar Java virtual machine benefits are all provided for embedded applications.

Because of Java SE and SE Embedded compatibility, and the fact that all Oracle Java Embedded Suite components are 100% Java code, developing embedded applications is considerably simpler than in languages such as C. Familiar Java development tools, such as the NetBeans integrated development environment, can be used. No cross-development tools are necessary, and there is no question of toolchain compatibility. Application bytecodes compiled on a desktop system run identically (except for speed) when copied to a target device. See Chapter 3 for more information on application development with Oracle Java Embedded Suite.

Chapter 4 gives more information on Oracle Java SE Embedded.

1.3 About Java DB

Java DB is a relational database accessed via SQL over JDBC. The Java DB edition included in Oracle Java Embedded Suite has been optimized for embedded applications by removing unneeded features such as the network client and server. For embedded applications, Java DB embeds in the application as a library.

Chapter 5 gives more information on Java DB.

1.4 About GlassFish for Oracle Java Embedded Suite

Oracle Java Embedded Suite includes a small-footprint version of the GlassFish application server. It supports hosting Servlet 3.0 applications. The GlassFish application server is packaged as a library that the virtual machine loads with an embedded application.

Chapter Chapter 6 gives more information on GlassFish.

1.5 About Jersey RESTful Web Services

Jersey is Oracle's implementation of JAX-RS (JSR 311), which is the Java API for RESTful web services. Jersey also includes additional features such as a proprietary RESTful client API. The small footprint lightweight HTTP server included in Oracle Java SE Embedded can host simple RESTful web services. Full-featured RESTful services can be hosted on the GlassFish container.

Chapter 7 gives more information on Jersey.

2

Installing and Verifying

This chapter describes how to install, deploy, and verify the Oracle Java Embedded Suite.

Table of Contents

2.1 Oracle Java Embedded Suite Bundles

The Oracle Java Embedded Suite is distributed in multiple zip file bundles.

- Code bundles: There are separate bundles for the x86/Linux and ARM/Linux target platforms.
- Samples bundle: The code in the samples bundle is 100% Java and runs on all target platforms.
- Documentation bundle: The documentation is provided in HTML and PDF formats. The top level `index.html` file links to the documents.

2.2 Installing the Bundles

To install the code and samples bundles on a host computer, follow these steps:

1. Create a directory, *installDir*, and change to it.
2. Download the code bundle to the current directory.
3. Unzip the code bundle, which creates *installDir*`/jes7.0/`. In *installDir*`/jes7.0/`, the file `jes-verify.sh` is for diagnostic use by Oracle Support. The other files and directories are described in this guide.
4. Download the samples bundle to the current directory (*installDir*).
5. Unzip the samples bundle, which creates *installDir*`/jes7.0/samples/`.

2.3 Deploying the Installation

To deploy an installation to the target device, copy *installDir* to the device by your choice of methods, such as a network connection or SD memory card. In this section, *deployDir* refers to the target directory containing `jes7.0/`.

To save space in the target device's file system, you can remove the samples:

```
$ rm -rf deployDir/jes7.0/samples
```

2.4 Verifying the Deployment

To verify the deployment, run `java -version` on the target device:

```
$ cd deployDir/jes7.0/jre/bin
$ ./java -version
java version "1.7.0_06"
Java(TM) SE Embedded Runtime Environment (build 1.7.0_06-b24, headless)
Java HotSpot(TM) Embedded Client VM (build 23.2-b09, mixed mode)
$
```

Some output details, such as build numbers, might differ.

For further verification, you can set the `JAVA_HOME` environment variable to *deployDir*`/jes7.0/jre`, then run the Java DB `sysinfo` command (partial output shown, details might differ):

```
$ cd deployDir/jes7.0/javadb/bin
$ ./sysinfo
------------------ Java Information ------------------
Java Version:    1.7.0_06
Java Vendor:     Oracle Corporation
Java home:       /home/user/jesB7/jes7.0/jre
Java classpath:
deployDir/jes7.0/javadb/lib/derby.jar:deployDir/jes7.0/javadb/lib/derbynet.jar:
deployDir/jes7.0/javadb/lib/derbytools.jar:deployDir/jes7.0/javadb/lib/derbyclient
.jar

...

--------- Derby Information --------
JRE - JDBC: Java SE 7 - JDBC 4.0
[deployDir/jes7.0/javadb/lib/derby.jar] 10.8.2.2 - (1181258)
[deployDir/jes7.0/javadb/lib/derbytools.jar] 10.8.2.2 - (1181258)
------------------------------------------------------
----------------- Locale Information -----------------
------------------------------------------------------
$
```

3

Application Development

This chapter gives an overview of developing applications with Oracle Java Embedded Suite. For details of working with each component, see Chapter 4, "Working with Oracle Java SE Embedded", Chapter 5, "Working with Java DB", Chapter 6, "Working with GlassFish", and Chapter 7, "Working with Jersey".

Table of Contents

3.1 Application Development Overview

Embedded application development is fundamentally cross-platform work. The target device (platform), on which the application will ultimately be deployed, usually does not have the hardware resources to support advanced development tools. Therefore, you build on a larger host computer, copy the required build artifacts to the target, and debug and tune the application running on the target. Your development environment might include remote execution tools that support debugging and profiling (monitoring memory use) from the host computer.

The method of copying files from the host to the target is highly target- and environment-specific. Options include physically moving a memory card between devices, using `scp`, and remotely mounting one computer's file system on the other.

3.2 Hybrid Development

The Oracle Java Embedded Suite's JRE is a subset of the full Standard Edition JRE APIs that are included in the JDK. The Java DB, GlassFish, and Jersey components are 100% Java code. Therefore, any application that can run on the target device can run on the host. The host's Java development tools are extensive and integrated. For example, on the host, you can write, run, debug, and profile an application in the NetBeans integrated development environment. There is no need for the target device. You must be sure, however, not to use a JDK API that is not present in the Oracle Java SE Embedded JRE.

3.3 Deploying an Application

You can copy a compiled application's files to any location in the embedded device's file system.

You must launch the application with the Oracle Java Embedded Suite launcher, which by default is *deployDir*`/jes7.0/jre/bin/java`. (NetBeans does not support remote launching of Oracle Java Embedded Suite applications.) You must specify the JAR files representing the components the application uses in the launcher's `-classpath` option. The component chapters in this guide list the JAR files, and you can see sample launch scripts in *installDir*`/jes7.0/samples/dist/run/` (see Chapter 8).

3.4 Modifying Samples

If a sample (see Chapter 8) is usefully close to the design of your application you can clone the `$JES_HOME/samples/` directory, delete unwanted samples, and build the desired sample with JDK command line tools or the NetBeans IDE, or an IDE of your choice. Examine the Ant scripts and the launch scripts for hints on building and running an application.

4

Working with Oracle Java SE Embedded

This chapter describes the relationship between Oracle Java SE Embedded and Java Standard Edition, and gives documentation sources.

The Java runtime environment included with Oracle Java Embedded Suite is the headless version of Java Standard Edition for Embedded Devices, which is a subset of Java Standard Edition. The main documentation pages for both products are:

- Java Standard Edition for Embedded Devices documentation: `http://www.oracle.com/technetwork/java/embedded/resources/se-embeddocs/index.html`
- Java Standard Edition developer documentation: `http://docs.oracle.com/javase/7/docs/index.html`
- Java Standard Edition tutorials: `http://docs.oracle.com/javase/tutorial/`

Ignore documentation of these Java Standard Edition features:

- The `-server` option of the `java` launcher command. The Oracle Java Embedded Suite `java` command implicitly uses the equivalent of the `-client` option.
- All graphics APIs and tutorials, in particular, those that mention the `java.awt` or `javax.swing` or `javafx` APIs. The Java Standard Edition for Embedded Devices software included with Java Embedded Suite does not support target device graphics.

5 Working with Java DB

This chapter describes the Oracle Java Embedded Suite's Java DB component.

Table of Contents

5.1 Java DB Features

The edition of Java DB included in Oracle Java Embedded Suite has all Java DB features except those that relate to Java DB's client/server environment. Client/server mode is not supported by the Java DB software included in Oracle Java Embedded Suite. The Java DB code is a library (`derby.jar`) that the JRE loads with the application.

5.2 Java DB Files

The Java DB files are in *installDir*`/jes7.0/javadb/`.

5.2.1 JAR Files

The following files are in the `javadb/lib/` directory:

- `derby.jar`: Always required on the compile and runtime classpaths
- `derbytools.jar`: Required on the compile and runtime classpaths if your application uses the Java DB tools described in Section 5.2.2

5.2.2 Binary Files

The following files are in the `javadb/bin/` directory:

- `dblook`: Displays the schema of a database.
- `ij`: Runs scripts or interactive queries against a database.
- `setEmbeddedCP`: Adds the `derby.jar` and `derbytools.jar` files to the classpath.
- `sysinfo`: Displays information about Java DB and the Java environment.

5.2.3 Obtaining Localized Error Message Files

If you want to present Java DB error messages to device end users, you can obtain non-English localizations of the messages from the Apache Derby `${javaDBversion}` download page, where `${javaDBversion}` is the Java DB version obtained from the Oracle Java Embedded Suite *Release Notes*.

The Apache Derby download page can be reached from `http://db.apache.org/derby/derby_downloads.html`. Download and unzip `db-derby-${javaDBversion}-lib.zip`, then look for the localization you need in `db-derby-${javaDBversion}-lib/lib`. Each localization is a JAR file named `derbyLocale_${locale}.jar` where `${locale}` is a supported language. Add the localization JAR file into your classpath.

5.3 Java DB Documentation

The Java DB documentation page is `http://docs.oracle.com/javadb/`. The *Oracle Java Embedded Suite Release Notes* give the relevant Java DB version number.

All Java DB documentation topics are relevant in the context of Oracle Java Embedded Suite, except for those related to the Java DB client/server environment. On an embedded device, the database and application run in the same JVM. Java DB uses the term "embedded environment" for the self-contained operation supported by Oracle Java Embedded Suite's Java DB component.

Here is a brief guide to the Java DB documents:

- Getting Started with Java DB:
  - Ignore the "Installing and configuring Derby" chapter", except for the sections on running the `sysinfo`, `dblook`, and `ij` tools.
  - In the "Self-study tutorial for users new to Derby" chapter, ignore Activity 2 and Activity 4, which use the Derby client/server environment.
- Java DB Reference Manual: Ignore the following:
  - All topics that refer to the client/server usage, which Oracle Java Embedded Suite does not support.
  - Java EE Compliance - Oracle Java Embedded Suite does not support distributed transactions.
- Java DB Developer's Guide: Ignore these chapters:
  - Upgrades - Do not upgrade GlassFish or any other Oracle Java Embedded Suite component.
  - Deploying derby applications, Deployment issues section.
- Tuning Java DB: This guide can help you improve database performance.
- Java DB Server and Administration Guide: This guide is divided into two parts. Ignore the topics in the Derby Server part except for "Multiple-client features available in Derby". An embedded application can create multiple database connections. The Administration Guide topics are relevant to Oracle Java Embedded Suite.
- Java DB Tools and Utilities Guide: Tools and scripts for setting up and using a database. Run these tools on the target device.

- Java DB API Documentation: Javadoc descriptions of the `org.apache.derby.*` APIs. The following server packages are not provided in Oracle Java Embedded Suite:
  - `org.apache.derby.drda`
  - `org.apache.mbeans.drda`

5.4 Java DB Samples

The following samples incorporate Java DB:

- hellostorage, Section 8.4
- hellosuite, Section 8.9

5.5 Java DB Notes

For best performance, pre-create databases and save them as JAR files. Creating a database on the fly is an expensive operation that can introduce user-observable delays in an application. The hellostorage sample (Section 8.4) is an example. The first invocation, which creates the database, takes much longer than subsequent runs, which update the existing database.

6

Working with GlassFish

This chapter describes the GlassFish edition that is included in Oracle Java Embedded Suite.

Table of Contents

6.1 GlassFish Features

The GlassFish edition included in Oracle Java Embedded Suite is packaged as a library whose classes your application invokes. To reduce memory footprint, the Oracle Java Embedded Suite GlassFish edition's feature set is a subset of the GlassFish Web Profile edition, as shown in Table 6–1. Notice that some missing features, such as Java DB, are provided by other Oracle Java Embedded Suite components.

Table 6–1 GlassFish Feature Comparison

Feature	GlassFish Web Profile	GlassFish for Oracle Java Embedded Suite
Part of Java EE 6 Specification	Yes	No
EJB 3.1 Lite	Yes	No
JTA/JTS	Yes	No
OSGI-based microkernel	Yes	No
Servlet 3.0	Yes	Yes
JSTL 2.2	Yes	No
JSP	Yes	No[1]
JSF	Yes	No
EL	Yes	No
JDBC	Yes	No[2]
Java DB	Yes	No[3]
Web Administration Console and CLI	Yes	No

Table 6–1 (Cont.) GlassFish Feature Comparison

Feature	GlassFish Web Profile	GlassFish for Oracle Java Embedded Suite
JAX-RS	Yes	No[4]
EJB 3.1	Yes	No
Bean Validation 1.0	Yes	Yes
Clustering	Yes	No
CDI	Yes	No
JPA 2.0	Yes	No
WAR Deployment	Yes	No

[1] JSP does not work in Oracle Java Embedded Client because it requires a Java compiler which the Java Runtime Environment does not have.
[2] JDBC is provided with the Java DB component.
[3] Java DB is a separate component of Oracle Java Embedded Suite.
[4] JAX-RS support is provided by Oracle Java Embedded Suite's Jersey component.

6.2 GlassFish Files

The GlassFish files are in *installDir*`/jes7.0/glassfish/`. The JAR file to include in the compile- and run-time classpaths is *installDir*`/jes7.0/glassfish/lib/glassfish-jes.jar`.

6.3 GlassFish Documentation

The GlassFish documentation page is `http://docs.oracle.com/cd/E18930_01/`.

Because GlassFish for Java Embedded Suite is a small subset of the complete GlassFish product, much of the documentation is not relevant. The most relevant document is the *Embedded Server Guide* (`http://docs.oracle.com/cd/E18930_01/html/821-2424/index.html`). Focus your attention on the topics describing web applications and servlets. The corresponding Javadoc is `http://embedded-glassfish.java.net/nonav/apidocs/`.

In the main Enterprise Edition Javadoc (`http://docs.oracle.com/javaee/6/api/`), the only relevant package is `javax.servlet`, except for `javax.servlet.jsp` and its sub-packages.

For information on securing access to web pages and services, consult these documents:

- *JAAS LoginModule Developer's Guide* (`http://download.java.net/jdk8/docs/technotes/guides/security/jaas/JAASLMDevGuide.html`)
- *The JAAS Reference Guide* (`http://docs.oracle.com/javase/7/docs/technotes/guides/security/jaas/JAASRefGuide.html`)

6.4 GlassFish Samples

The following samples incorporate GlassFish:

- webhost, Section 8.5
- helloservice, Section 8.7
- helloservlet, Section 8.6
- hellosuite, Section 8.9

7

Working with Jersey

This chapter describes the essentials of working with Oracle Java Embedded Suite's Jersey component.

Table of Contents

7.1 Jersey Features

Oracle Java Embedded Suite includes Jersey 1.11.1 server, client, core, JSON, and ask.

> **Note:** The Jersey client API is proprietary and is likely to be superseded by a future standard client API defined in JAX-RS 2.0.

The Jersey edition included with Oracle Java Embedded Suite does not have the following features:

- Support for multipart/* media type
- Support for OAuth
- Support for Atom
- MVC using freemarker templating
- Support for FastInfoset
- Declarative server hyperlinking support
- Non-blocking client implementation
- Integration with Apache HTTP Client
- Integration with Guice and Spring
- Integration with MOXy (EclipseLink JAXB implementation)

7.2 Jersey Files

The Jersey JAR files are in *installDir*`/jes7.0/jersey/lib/`.

- Core function files
  - asm.jar
    - Description: Third-party library required by jersey-server.
    - Dependencies: (none)
  - jersey-core.jar
    - Description: Jersey core, required by both client and server
    - Dependencies: (none)
  - jersey-server.jar
    - Description: Jersey server library. Sufficient for running Jersey on lightweight HTTP server.
    - Dependencies: asm.jar, jersey-core.jar, JAXB from JDK for XML <-> Java binding support.
- jersey-client.jar
  - Description: Client support file
  - Dependencies: jersey-server.jar, jersey-core.jar
- jsr311-api.jar
  - Description: Defines the RESTful services API; required to compile a RESTful server or client.
  - Dependencies: None
- jersey-servlet.jar
  - Description: Jersey servlet module
  - Dependencies: jersey-server.jar
- jersey-json.jar
  - Description: JSON support module.
  - Dependencies: jersey-core.jar

7.3 Jersey Documentation

For information on using Jersey, refer to the following documents:

- Jersey User Guide: Overview of and tutorial for Jersey. The samples in this guide assume the use of Maven for automatically downloading dependencies. If Maven is not available, you can specify the JARs included with JES on the compiler and runtime classpaths. http://jersey.java.net/nonav/documentation/latest/user-guide.html#chapter_deps specifies JAR dependencies for the classpath.

  Link:
  http://jersey.java.net/nonav/documentation/1.11.1/index.html
- APIs (Javadoc): Standard API documentation for com.sun.jersey.*, javax.ws.rs.*, com.sun.ws.rs.ext.

  Link:
  http://jersey.java.net/nonav/apidocs/1.11/jersey/index.html

Ignore the information on this page:
https://wikis.oracle.com/display/Jersey/Main.

7.4 Jersey Samples

The following samples incorporate Jersey:

- helloservice, Section 8.7
- helloclient, Section 8.8
- hellosuite, Section 8.9

8 Samples

This is a chapter describes the code samples that are available with Oracle Java Embedded Suite.

Table of Contents

8.1 About the Samples

The samples are deliberately very simple in scope, analogous to the Hello, World examples written for many software products. They illustrate basic use of each component without the complication of doing useful application work. For example, the hellostorage sample (Section 8.4) creates a Java DB database of timestamps, adding one row each time it is invoked.

In addition to using the sample source code to learn about using the components, you can use a sample as a template for your own application, progressively modifying it to meet your application requirements.

If you are evaluating Oracle Java Embedded Suite, reviewing, and optionally running, the samples can give you a quick experience of the product's basic capabilities.

8.2 Sample Installation and Prerequisites

To read the samples code, you must install the Oracle Java Embedded Suite and its samples bundle as described in the *Chapter 2*. The most convenient way to explore the samples is as NetBeans projects, but you can also open the files with a text editor.

Note: This chapter uses the notation `$JES_HOME` as shorthand for *installDir*`/jes7.0/`. The sample scripts automatically set and use the environment variable of the same name.

8.3 Sample Directories and Structure

The samples share a directory and have similar subdirectory structures.

8.3.1 samples Directory

The samples are located in `$JES_HOME/samples/`. This directory contains:

- `scripts/`: Source code for bash shell scripts to run each sample, for example, `hclient.sh` for the HelloClient sample. These scripts take care of setting the classpath to include the required components and invoking the sample. They can be used as models for your own launch scripts. Do not run the scripts in this directory. Run the scripts in `dist/run/`.
- `build.xml`: An umbrella Ant script that operates over all samples. Its targets are `clean` and `build`.
- dist/: Created by the umbrella Ant script, `dist/` contains application jar and war files, and the `run/` directory from which you can invoke sample scripts.

8.3.2 Common Sample Structure

Each sample directory has a similar structure. The top directories and files are:

- `nbproject/`: NetBeans project definition files and `build-impl.xml`, which is build specifications used by Ant and NetBeans.
- `build/`: Build artifacts that can be ignored.
- `build.xml`: Command line Ant build script. It imports `nbproject/build-impl.xml` which is the detailed build specifications. Editing `build.xml` is a convenient way to specify values for placeholder targets defined in `nbproject/build-impl.xml`.
- dist/: Created by the build, contains the files for running the sample scripts.
- `src/`: Source files contained in `java/` and, for some samples, `webapp/` subdirectories.

8.3.3 Running Samples

You must run samples from `$JES_HOME/samples/dist/run/`. Later sections of this chapter give details for each sample. You cannot run the samples in NetBeans.

In `$JES_HOME/samples/dist/run/`, the file `config.sh` defines environment variables that other scripts use. In particular, it defines classpaths for various combinations of components. This technique enables scripts to specify classpaths that minimize the runtime memory footprint by including only the components the script needs. For instance, a script that runs the JRE lightweight HTTP server does not need the GlassFish component.

`config.sh` also sets the runtime heap size to 64 megabytes (`-Xmx64m` option to the `java` command). This value is sufficient for the samples as they are defined, but it might limit your execution of samples with your own inputs. For example, specifying several large web applications to `gfhost.sh` (described in Section 8.5.3) might exhaust the

default 64 megabyte heap. If the target device has more or less space for the heap, you can modify `config.sh` accordingly. When you run the samples on the host computer, you will get an `OutOfMemoryError` if you attempt to use more heap than the target device has.

To run samples on the target device, create a `jes7.0/` directory on the target, then recursively copy `$JES_HOME/samples/dist/` to the target's `jes7.0/`.

8.4 hellostorage Sample

This sample creates, updates, and displays a persistent database of timestamps. It adds a row each time `main()` is invoked.

8.4.1 Components

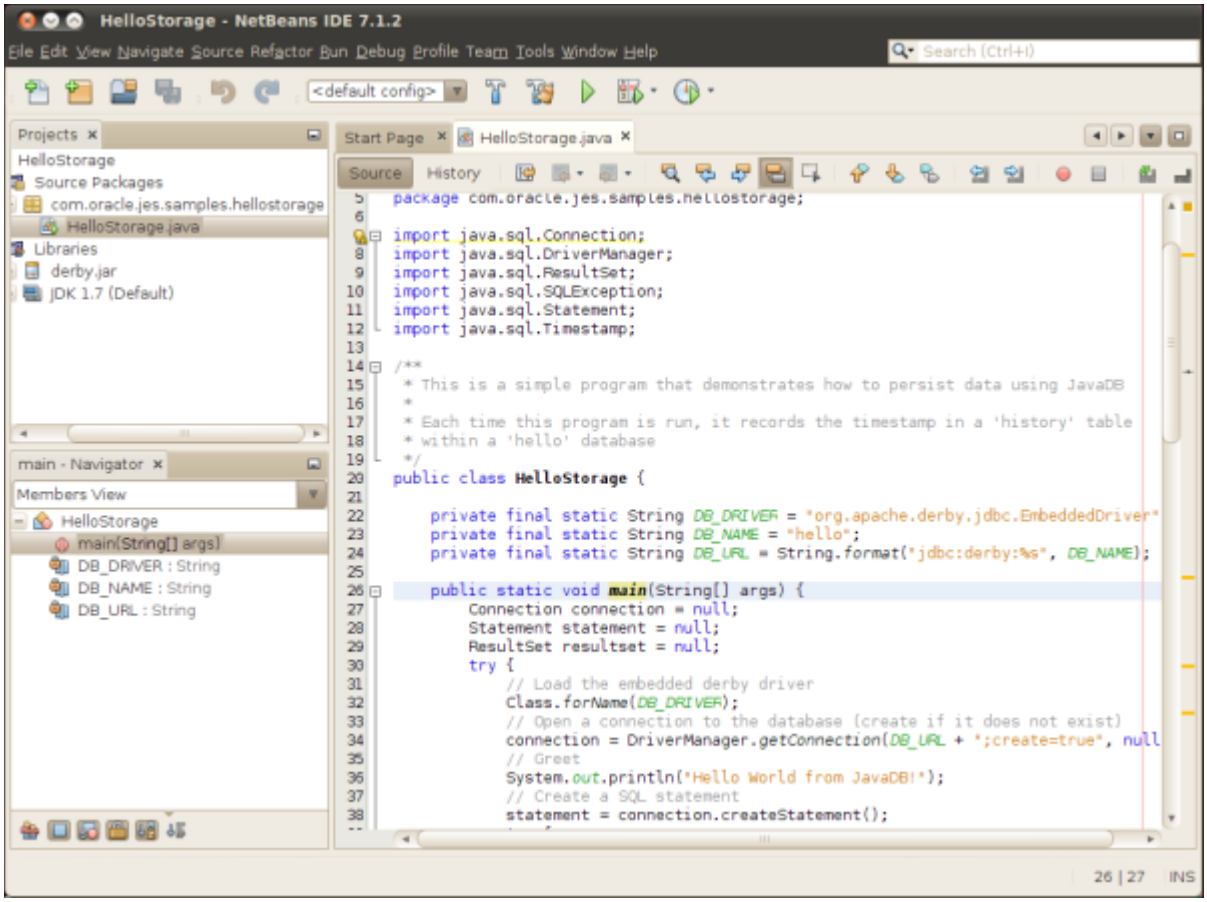
JRE, Java DB (embedded JDBC driver only, see Section 8.4.4)

8.4.2 Source Files

Location: `$JES_HOME/samples/hellostorage/`

Description: `HelloStorage.java` is the complete sample. See Figure 8–1 for an overview.

Figure 8–1 Overview of hellostorage Sample Files



In more detail, the program:

- Loads the Embedded Driver from JavaDB.
- Creates a JDBC connection for 'hello' database.
- If this is the first run, creates a 'history' table which contains id & timestamp columns.
- Inserts the current timestamp into the history table (id is auto-generated).
- Queries the history table & prints all the contents indicating the list of timestamps when this program was run (sorted by id).
- Closes the JDBC connections.
- Shuts down the database.

8.4.3 Run Instructions and Sample Output

Bold type indicates user input.

```
$ cd $JES_HOME/samples/dist/run
$ ./hstorage.sh
Hello World from JavaDB!
    Hello #1 @ 2012-05-10 11:21:17.164
$ ./hstorage.sh
Hello World from JavaDB!
    Hello #1 @ 2012-05-10 11:21:17.164
    Hello #2 @ 2012-05-10 11:21:25.988
$
```

8.4.4 Notes

- The sample does not use Java DB APIs, and can therefore be compiled without `derby.jar` in the classpath. However, `derby.jar` must be specified in the runtime classpath because it supplies the embedded JDBC driver.
- The first run, which creates the database, can take on the order of 10 seconds to complete. Subsequent runs are faster.
- The database is created in the directory in which you launch the sample. The database files are stored in a subdirectory named `hello/`. You can inspect the database with `$JES_HOME/javadb/bin/ij`.
- You can delete the database with `rm -rf hello`.

8.5 webhost Sample

The webhost sample is a utility that provides common interfaces for running applications on either GlassFish or the JRE's lightweight HTTP server. It implements a management lifecycle, starting the server, deploying one or more applications, undeploying applications, and stopping the server.

If you have existing web applications that run on standalone application servers, the webhost utility can help you get them running on the embedded GlassFish server included with Oracle Java Embedded Suite.

If you have annotation-based JAX-RS (Jersey) applications, you can use webhost to deploy them on the lightweight HTTP server. See the HelloService (Section 8.7) and hellosuite (Section 8.9) samples for examples of Jersey applications supported by

webhost. The lightweight HTTP server's memory footprint is much smaller than GlassFish's.

8.5.1 Components

Jersey, GlassFish, JRE (notably `com.sun.net.httpserver`)

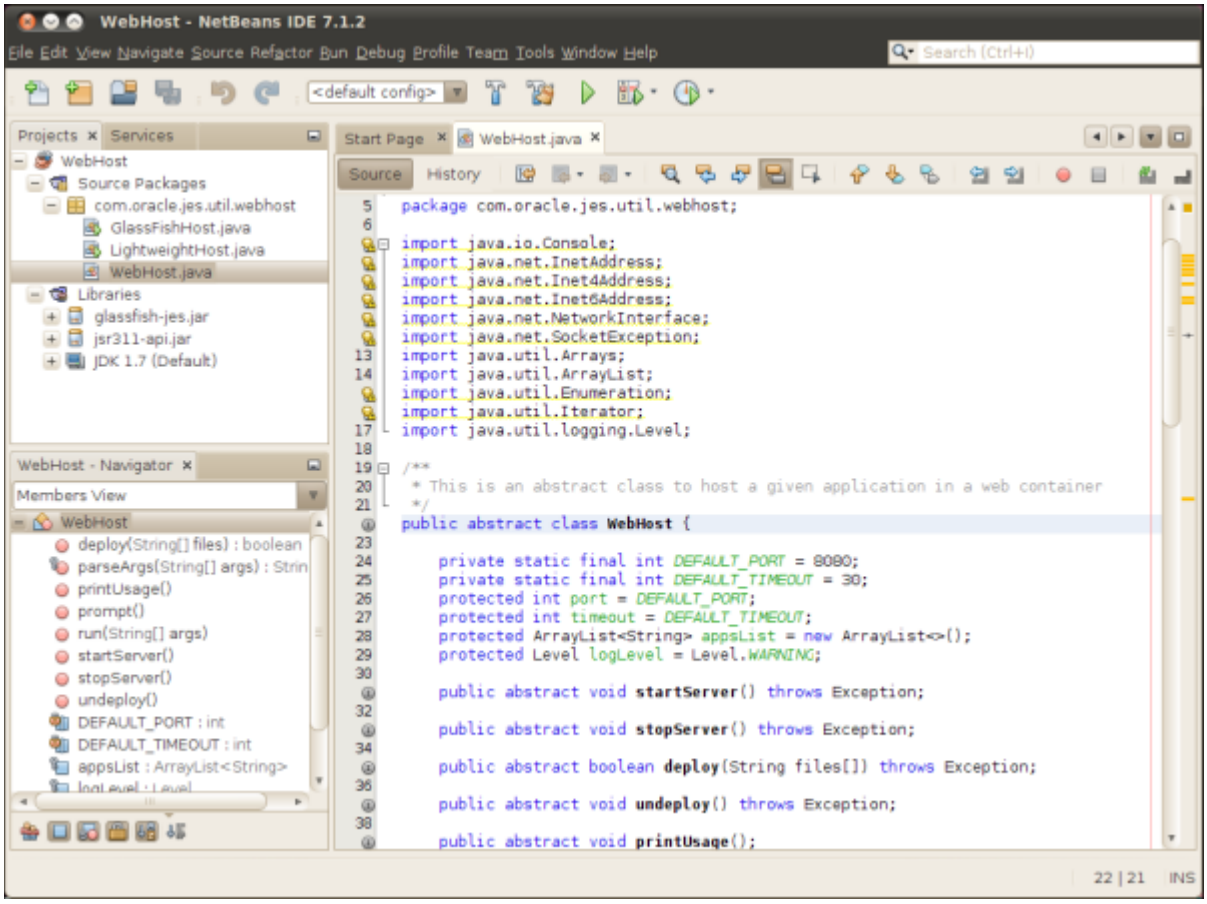
8.5.2 Source Files

Location: `$JES_HOME/samples/webhost/`

Descriptions: See Figure 8–2 for an overview.

- `WebHost.java`: An abstract class that defines an interface to, and partial implementation of, a generic webhost.
- `GlassFishHost.java`: Implements a webhost with the embedded GlassFish application server (`org.glassfish.embeddable` APIs) to deploy web applications packaged in WAR files. There are other ways to deploy a GlassFish application, but `GlassFishHost.java` does not support them.
- `LightweightHost.java`: Implements a webhost with the JRE lightweight HTTP server, deploying applications packaged in JAR files by creating a handler and context for each. It supports only annotation-based JAX-RS applications, but could be modified to support other styles of Jersey applications.

Figure 8–2 Overview of webhost Sample Files



8.5.3 Run Instructions

The scripts `gfhost.sh` and `lwhost.sh`, respectively, launch the GlassFish and lightweight webhosts. For a lightweight webhost example, see Section 8.7.3. For the GlassFish webhost, see Section 8.6.3.

8.5.4 Notes

- To compile the webhost sample, specify `glassfish-jes.jar` in the classpath.
- To run the GlassFish webhost, include `glassfish-jes.jar` in the classpath (`gfhost.sh` does this automatically). The lightweight webhost has no classpath requirement.

8.6 helloservlet Sample

helloservlet illustrates the Servlet 3.0 application lifecycle, running in a GlassFish container. (The lightweight HTTP server does not support servlets.)

8.6.1 Components

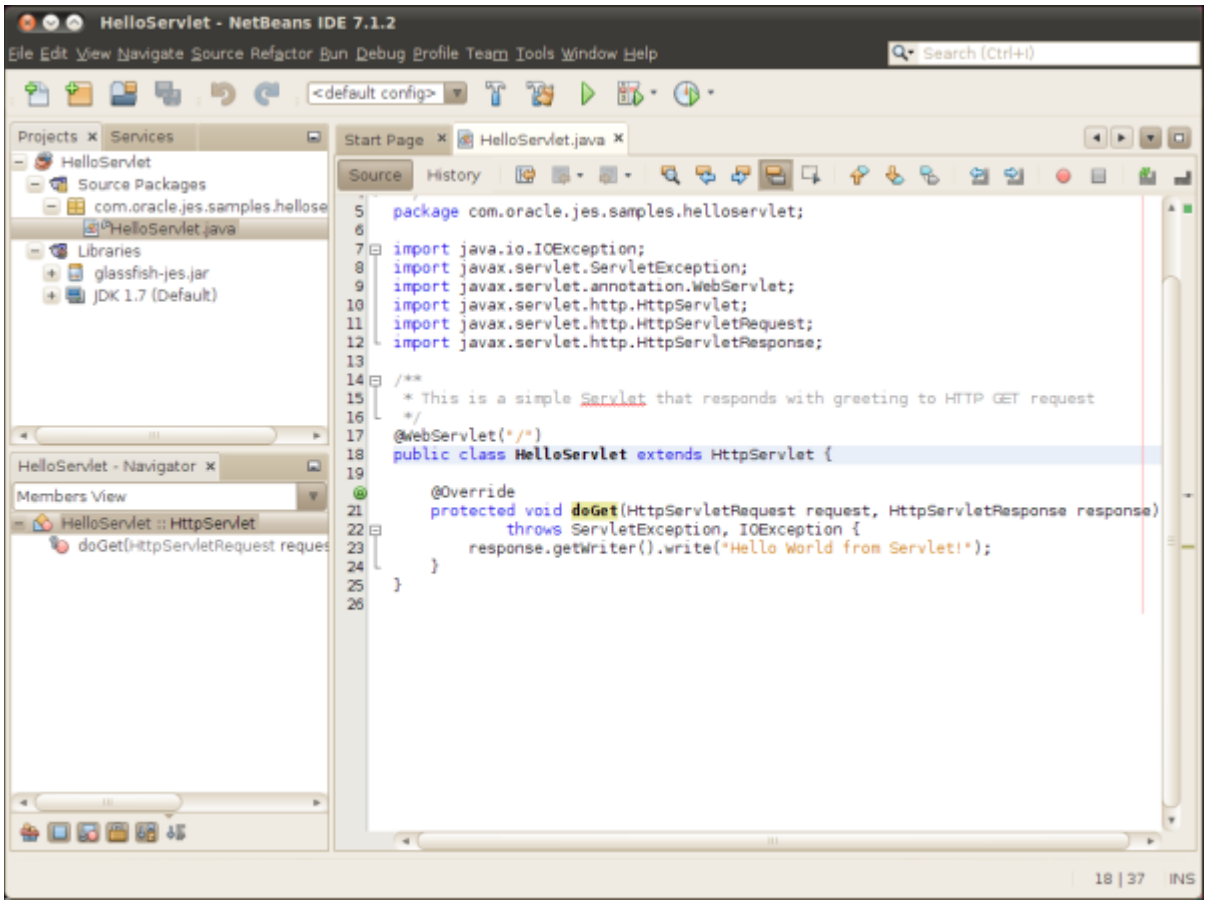
GlassFish, JRE

8.6.2 Source Files

Location: `$JES_HOME/samples/helloservlet/`

Description: For an overview, see Figure 8–3. The code is very simple, defining the context for the servlet ("/"), and a `doGet()` method that returns a string.

Figure 8–3 Overview of helloservlet Sample Files



8.6.3 Run Instructions and Example Output

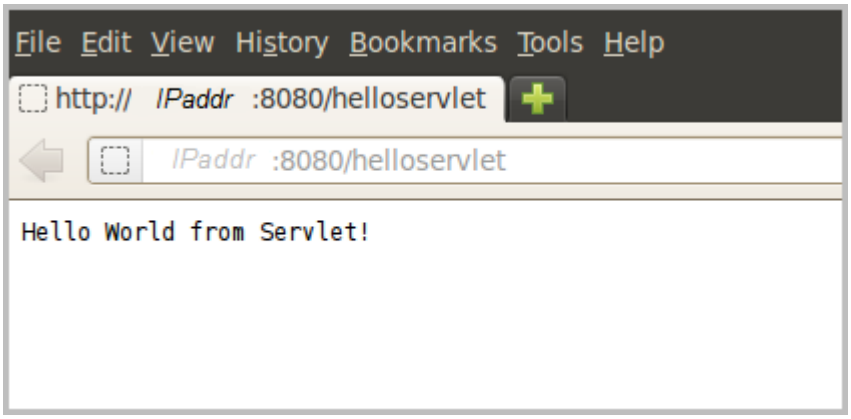
Bold type indicates user input.

1. Start the servlet in the GlassFish webhost with the `gfhost.sh` script (described in Section 8.5)

```
$ cd $JES_HOME/samples/dist/run
$  ./gfhost.sh ../helloservlet.war
Preparing to run glassfish host...
May 11, 2012 1:57:44 PM
com.sun.enterprise.v3.server.CommonClassLoaderServiceImpl findDerbyClient
INFO: Cannot find javadb client jar file, derby jdbc driver will not be
available by default.
May 11, 2012 1:57:45 PM org.hibernate.validator.util.Version <clinit>
INFO: Hibernate Validator 4.2.0.Final
May 11, 2012 1:57:46 PM com.sun.enterprise.v3.services.impl.GrizzlyService
createNetworkProxy
...
Here is the list of installed apps:
    helloservlet
Please visit following URL(s)to verify app(s)
    http://IPaddr:8080/<ROOT>/[<URI>]
Please refer to app docs for the context ROOT & URI
Press <Enter> to exit server
```

2. In a web browser, enter the IP address given near the end of the output with the web application name, for example, http://*IPaddr*:8080/helloservlet. The servlet response is similar to Figure 8–4.

Figure 8–4 Output from helloservlet Sample



3. Stop the servlet by pressing <Enter> in the shell in which you launched `gfhost.sh`.

```
Undeploying: helloservlet
PlainTextActionReporterSUCCESSNo monitoring data to report.
Stopping server
com.sun.enterprise.glassfish.bootstrap.StaticGlassFishRuntime$1@1b6d9cf
May 11, 2012 2:27:52 PM com.sun.enterprise.v3.server.AppServerStartup stop
INFO: Shutdown procedure finished
May 11, 2012 2:27:52 PM AppServerStartup run
INFO: [Thread[GlassFish Kernel Main Thread,5,main]] exiting
```

8.6.4 Notes

Unlike the RESTful service described in Section 8.7, helloservlet is not accessible from a Java program, only a web browser.

8.7 helloservice Sample

helloservice is a simple RESTful service that returns HTML text to a web browser client, or a bean to a Java program client. The service can be deployed on the GlassFish or lightweight webhost (see Section 8.5).

8.7.1 Components

Jersey and either GlassFish or JRE lightweight HTTP server

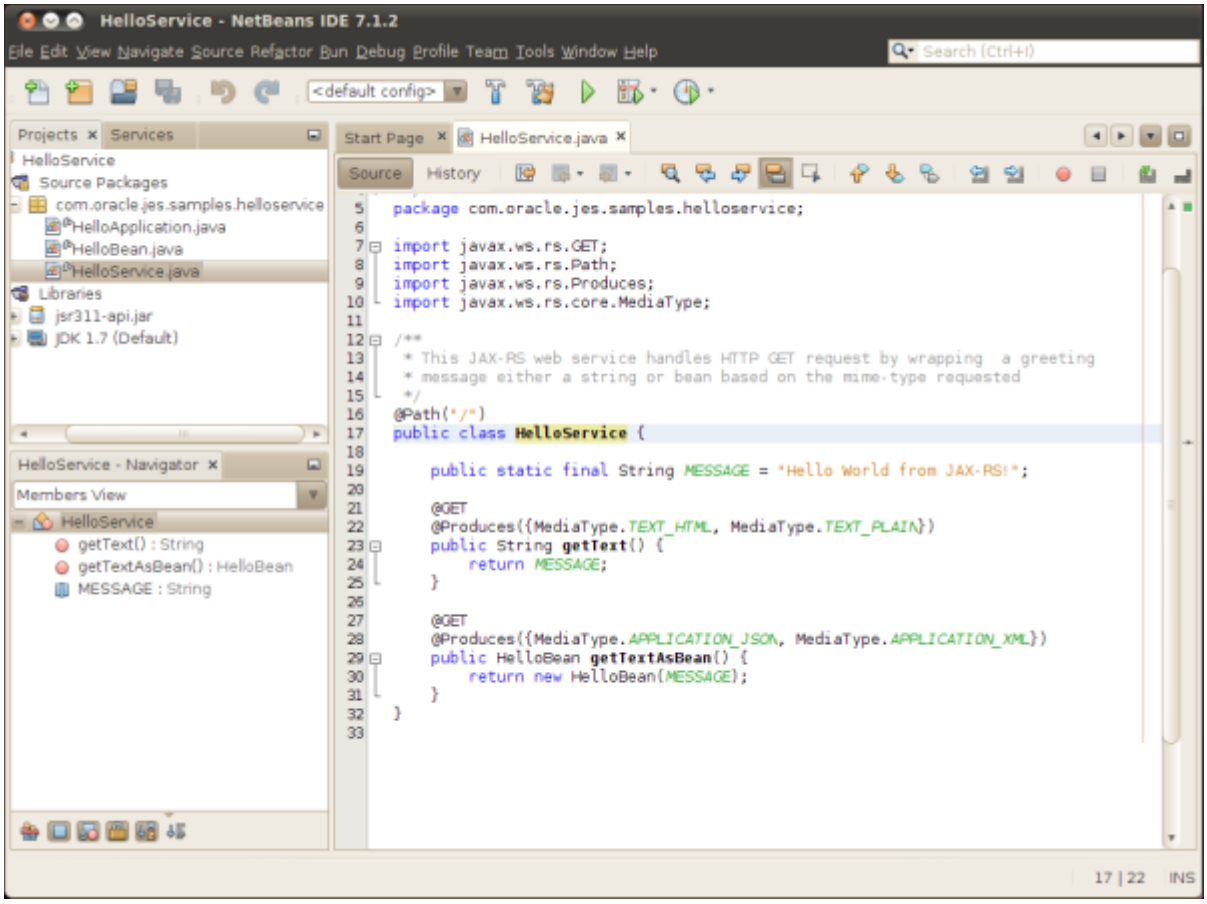
8.7.2 Source Files

Location: `$JES_HOME/samples/helloservice/`

Descriptions: See Figure 8–5 for an overview.

- `HelloApplication.java`: Extends the JAX-RS `Application` class whose single `getClasses()` method defines the application resources, in this case, the `HelloService` class.
- `HelloService.java`: The service's main handler, it defines two JAX-RS GET operations. One returns HTML text to browser clients. The other returns a Java object (a bean) to Java clients.
- `HelloBean.java`: This class implements the bean that HelloService returns to Java clients.

Figure 8–5 Overview of helloservice Sample Files



8.7.3 Run Instructions and Sample Output

The following instructions deploy the helloservice sample on the lightweight webhost, using `lwhost.sh`. In these instructions, you use two shells, one to launch the service, the other to run client invocations. You also invoke the service from a web browser.

Bold type indicates user input.

1. Start the lightweight webhost running the helloservice RESTful service:

```
$ cd $JES_HOME/samples/dist/run
$ ./lwhost.sh ../helloservice.jar
com.oracle.jes.samples.helloservice.HelloApplication
```

```
Preparing to run lightweight host...
Deploying: ../helloservice.jar
May 11, 2012 3:25:50 PM
com.sun.jersey.server.impl.application.WebApplicationImpl _initiate
INFO: Initiating Jersey application, version 'Jersey: 1.11.1 03/31/2012 06:49
PM'
Here is the list of installed apps:
    helloservice
Please visit following URL(s)to verify app(s)
    http://IPaddr:8080/<ROOT>/[<URI>]
Please refer to app docs for the context ROOT & URI
Press <Enter> to exit server.
```

To deploy on the GlassFish webhost, substitute **`./gfhost.sh ../helloservice.war`** for **`./lwhost.sh ../helloservice.jar`**.

2. Invoke the service from a web browser client using the IP address given near the end of the `lwhost` output, for example, http://*IPaddr*:8080/helloservice. The service responds similarly to Figure 8–6.

Figure 8–6 Browser Client Output from helloservice Sample

3. Stop the service by pressing <Enter>.

```
undeploying: helloservice
```

8.8 helloclient Sample

The helloclient sample creates a RESTful client and invokes helloservice (described in Section 8.7) as specified by `hclient.sh`'s two arguments, type and URL. The type argument values are "text," "html", "xml", and "json" (all lower case or upper case). The URL argument identifies helloservice.

There is an alternative "type", which gets a bean object from the service. For this case, the type is the name of the bean class, the URL identifies the service, and a third argument is the JAR file on the classpath that contains the bean class.

This client can also used with the hellosuite sample (see Section 8.9).

8.8.1 Components

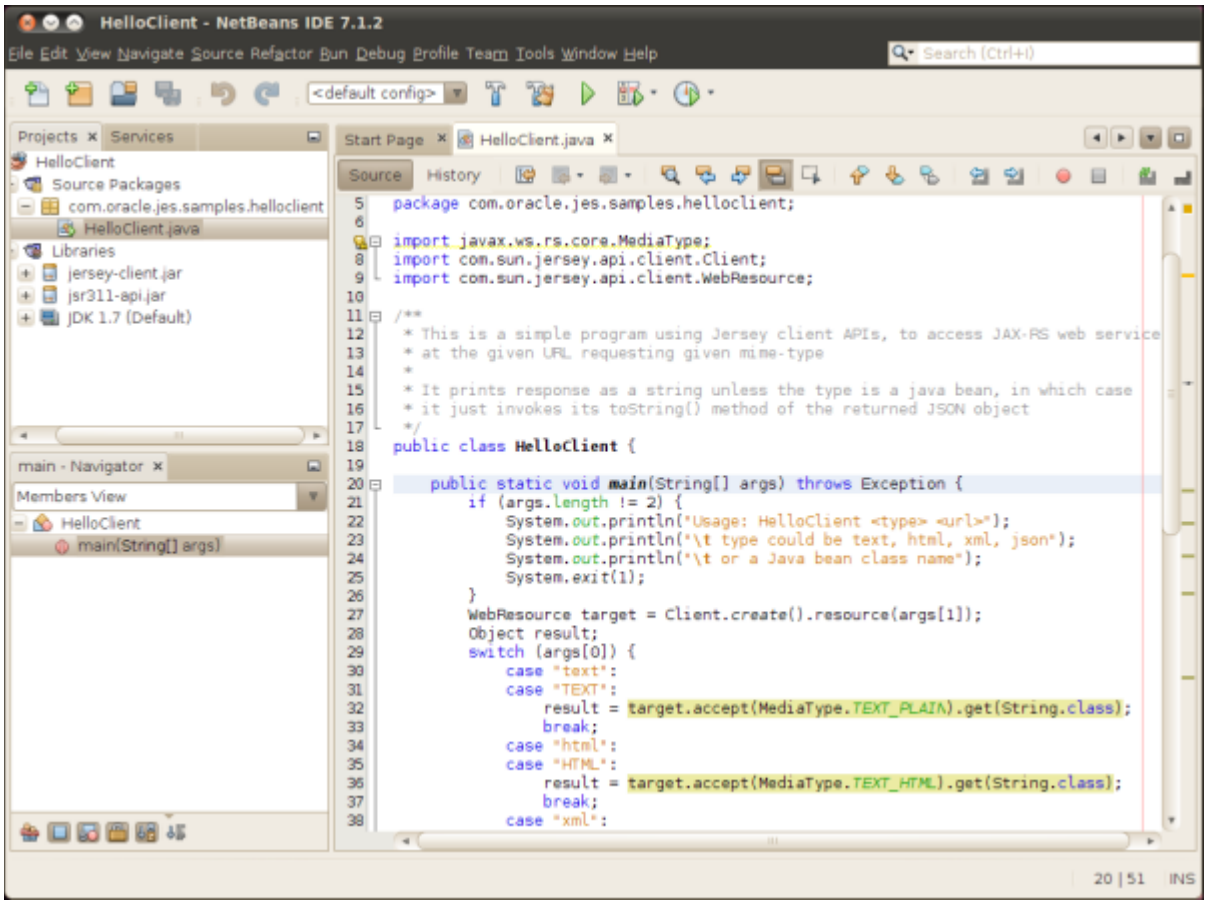
Jersey (client and server)

8.8.2 Source Files

Location: `$JES_HOME/samples/helloclient/`

Description: See Figure 8–7 for an overview. The single source file is `HelloClient.java`, which creates the `Client` and then executes a `switch` based on the type argument.

Figure 8–7 Overview of helloclient Sample Files



8.8.3 Run Instructions and Sample Output

These instructions demonstrate how to use helloclient to obtain a helloservice result in JSON format and as a bean object.

Bold type indicates user input.

1. Launch helloservice as described in Section 8.7.3.
2. In a second shell, launch helloclient specifying a type of JSON.

```
$ cd $JES_HOME/samples/dist/run
$ ./hclient.sh JSON http://IPaddr:8080/helloservice
{"text":"Hello World from JAX-RS!"}
```

3. In the second shell, launch helloclient, asking for the result in a `HelloBean` object.

```
$ cd $JES_HOME/samples/dist/run
$ ./hclient.sh com.oracle.jes.samples.helloservice.HelloBean
```

```
http://IPaddr:8080/helloservice ../helloservice.jar
[Hello World from JAX-RS!]
```

4. In the first shell, terminate helloservice by pressing <Enter>.

8.8.4 Notes

If you use the bean object option, the JAR containing the bean class must be included in the classpath. The `hclient.sh` script adds the third argument (`../helloservice.jar` in Section 8.8.3) to the classpath. To keep helloclient simple, the bean class is in `helloservice.jar`. A real application would likely define the bean in a separate JAR file included in both the client and server class paths.

8.9 hellosuite Sample

The hellosuite sample illustrates all of the Oracle Java Embedded Suite components used in combination. The sample is a Jersey RESTful service that runs in a GlassFish or lightweight HTTP container and stores data in a Java DB database.

8.9.1 Components

Jersey, Java DB, GlassFish, JRE (in particular, the lightweight HTTP server)

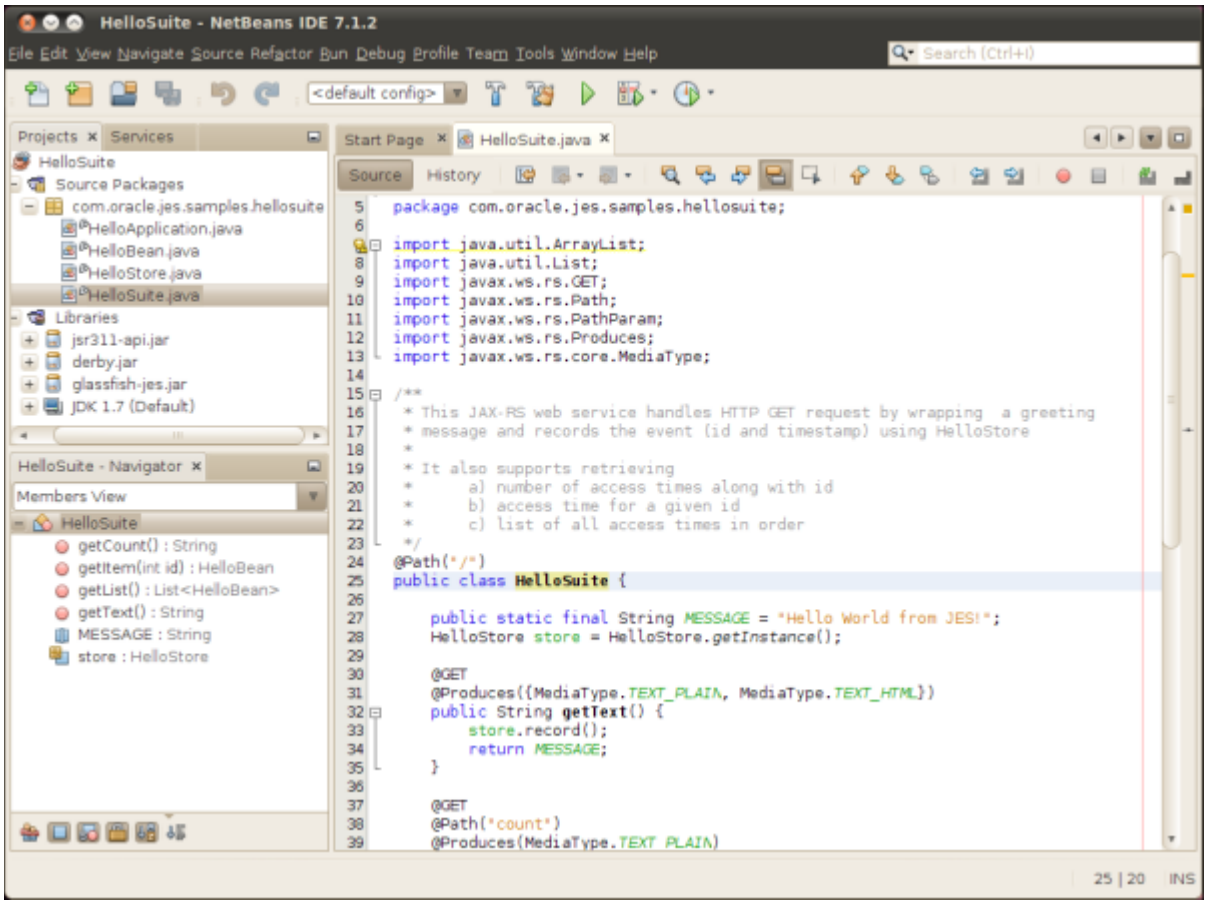
8.9.2 Source Files

Location: `$JES_HOME/samples/hellosuite/`

Description: See Figure 8–8 for an overview.

- `HelloApplication.java`: Extends the JAX-RS `Application` class whose single `getClasses()` method defines the application resources, in this case, the `HelloService` class.
- `HelloBean.java`: This class implements the bean that HelloSuite returns to Java clients.
- `HelloStore.java`: This class handles persistence with Java DB. It creates the database, adds timestamp rows, lists the contents of a particular row or all rows.
- `HelloSuite.java`: This is the service class. It defines these contexts for the service URL:
  - /: Adds a timestamp row to the database and returns "Hello world from JES!" as text or HTML.
  - /count: returns the number of database rows as text.
  - /list: lists the database records in XML or JSON format.
  - /*id*: Returns the database record indexed by *id* in XML or JSON format. The first record has an *id* of 1. This illustrates taking an input parameter from the client.

Figure 8–8 Overview of hellosuite Sample Files



8.9.3 Run Instructions and Sample Output

The following instructions use the GlassFish webhost. To use the lightweight webhost, see Section 8.9.4.

You use two shells in these instructions, the first to run the hellosuite service, the second to launch clients that use the service. If you have run this sample or hellostorage previously, you might want to remove the old database before running this sample. Existing database timestamps are potentially confusing. Remove the old database as follows:

```
$ cd $JES_HOME/samples/dist/run
$ rm -rf hello
```

Bold type indicates user input.

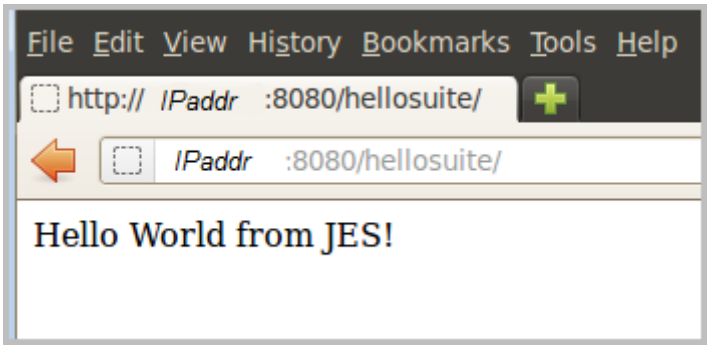
1. In a shell, start the hellosuite service:

```
$ cd $JES_HOME/samples/dist/run
$ ./gfhost.sh ../hellosuite.war
Preparing to run glassfish host...
May 17, 2012 1:47:08 PM
com.sun.enterprise.v3.server.CommonClassLoaderServiceImpl findDerbyClient
...
INFO: hellosuite was successfully deployed in 4,006 milliseconds.
Here is the list of installed apps:
```

```
    hellosuite
Please visit following URL(s)to verify app(s)
    http://IPaddr:8080/<ROOT>/[<URI>]
Please refer to app docs for the context ROOT & URI
Press <Enter> to exit server.
```

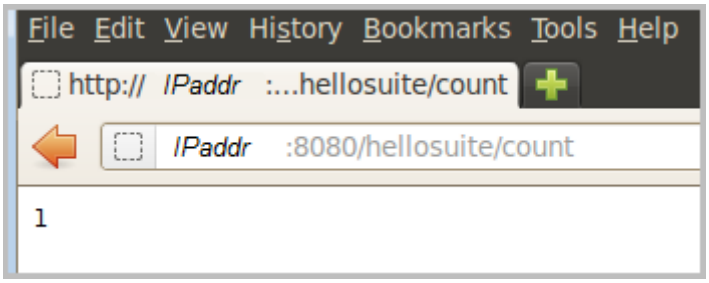
2. In a web browser, enter the IP address given near the end of the `gfhost` output (shown in italic type above) with the service name: http://*IPaddr*:8080/hellosuite. The service response is similar to Figure 8–9.

Figure 8–9 Browser Client Output for hellosuite Sample



3. In the web browser, enter a URL similar to http://*IPaddr*:8080/hellosuite/count. The browser displays a count of 1 similar to Figure 8–10.

Figure 8–10 Browser Client Output for hellosuite/count



4. Repeat step 2 a few times to direct the service to add timestamp rows to the database.
5. Repeat step 3 to get a count of the current number of rows in the database.
6. In the web browser, enter http://*IPaddr*:8080/hellosuite/list to see an XML list of the database records.
7. In the web browser, enter http://*IPaddr*:8080/hellosuite/3 to see an XML display of the third record.
8. In a second shell, run helloclient to lists the database records in JSON format. The results are similar to the following (ignore line breaks).

```
$ cd $JES_HOME/samples/dist/run
$ ./hclient JSON http://localhost:8080/hellosuite/list
{"helloBean":[{"id":"1","time":"2012-05-17
15:28:32.749"},{"id":"2","time":"2012-05-17
18:06:17.285"},{"id":"3","time":"2012-05-17
18:06:21.968"},{"id":"4","time":"2012-05-17 18:06:24.049"}]}
```

```
$
```

9. In the second shell run helloclient to list the first (or other) database record as a bean.

```
$ ./hclient.sh com.oracle.jes.samples.hellosuite.HelloBean
http://localhost:8080/hellosuite/1 ../hellosuite.jar
[ 1# 2012-05-17 15:28:32.749 ]
```

10. In the first shell, press <Enter> to terminate the service.

8.9.4 Notes

- The sample does not use JavaDB APIs, and can therefore be compiled without `derby.jar` in the classpath. However, `derby.jar` must be specified in the runtime classpath because it supplies the embedded JDBC driver.
- The first invocation, which creates the database, can take on the order of 10 seconds to complete. Subsequent runs are faster.
- The database is created in the directory in which you launch the sample. The database files are stored in a subdirectory named `hello/`. You can inspect the database with `$JES_HOME/javadb/bin/ij`.
- You can delete the database with `rm -rf hello`.
- You can run the service with the lightweight webhost instead of GlassFish, by making this step 1:

```
$ ./lwhost.sh ../hellosuite.jar
com.oracle.jes.samples.hellosuite.HelloApplication
```