Sun[tm] ONE Studio 4, Enterprise Edition for Java[tm] with Application Server 7 Tutorial

817-0233-10
October 2002

Contents

About This Guide 5

What You Should Know 5
How This Guide is Organized 5
Documentation Conventions 6

Chapter 1 Introducing Sun ONE Studio 4, Enterprise Edition for Java 7

Sun ONE Studio 4 Integration with Sun ONE Application Server 7 8

Chapter 2 Setting Up Your Development Environment 11

Sun ONE Studio 4 and Application Server Integration 11
  Installing the IDE Plugin 11
  Connecting the IDE to Application Server Deployments 12
Setting Up Your Development Environment 12
Install Sun ONE Studio 4, Enterprise Edition for Java 14
Install IDE Plugin 15
  Application Server and IDE Installed on Same Machine 15
    Download Plugin from the Sun ONE Studio Update Center 15
    Install the Plugin Using the Sun ONE Application Server 7 Installer 16
  Application Server Installed on Remote Development Server 17
Register Target Application Server 17
  Start the Application Server 18
    Start the Application Server on Windows Platforms 18
    Start the Application Server on UNIX Platforms 18
  Start the IDE 18
    Start the IDE on Windows Platforms 19

Start the IDE on UNIX Platforms 19
Set Sun ONE Application Server as the Default Server 20
Ensure Plugin is Configured with Application Server Installation Directory 20
Register a Sun ONE Application Server 21
Verify that the Application Server Instance is Running 24

Chapter 3 Developing Session Beans and Web Applications 25

Becoming Familiar with the Sample 26
Creating a Session Bean with the EJB[tm] Builder 29
Create the Session Bean 30
Modify JNDI Name 32
Modify Automatically Generated Methods 33
Create a Business Method 34
Testing the Session Bean 38
Create the Test Client 38
Deploy the Test Client 40
Run the Test Client 42
Packaging the Enterprise Bean in an EJB[tm] Module 47
Creating the Web Application 48
Create the Web Module 49
Create Servlets 51
Create GreeterDBServlet 52
Create GreeterDBLogDisplayServlet 59
Define Servlet Mappings 62
Create HTML and JSP Files 63
Define EJB Reference 64
Define JDBC Resource Reference 66
Assembling the J2EE[tm] Application 67
Setting Up Database Connectivity 70
Configure the JDBC Driver 70
Define the JDBC Connection Pool 71
To determine whether or not the connection pool has already been registered: 71
To register the JDBC connection pool: 71
Define the JDBC Resource 74
To determine whether or not a suitable JDBC resource has already been registered: 75
To register the JDBC resource (referred to as a JDBC "DataSource" in the IDE): 75
Start the PointBase Server Database 77
Deploying and Running the J2EE[tm] Application 78
Disable EJB Test Application 79
Define Web Context Root 79
Deploy the Application 81
Run the Application 81

Modifying the Application 83
Modify HTML File 83
Modify JavaServer Pages[tm] (JSP[tm]) File 84
Modify Servlet 84
Modify EJB 85
Debugging the Application 86
Prepare Web Module for Debugging 86
Start the Debugger 87
Debug the EJB 87
Debug a JSP 88
Stop the Debugger 89

Chapter 4 Summary 91

Index 93

About This Guide

This tutorial is intended to be used by first time users of the Sun Open Network Environment (Sun ONE) Application Server 7 using Sun ONE Studio 4, Enterprise Edition for Java (formerly Forte[tm] For Java, Enterprise Edition.)

The tutorial offers a hands-on means of gaining familiarity with development of basic J2EE[tm] applications using the Studio IDE. Completion of the exercies in this tutorial will typically take one to two hours of your time. Prior application server and development experience is not required in order to follow this tutorial.

See the Sun ONE Application Server 7 documentation collection on docs.sun.com for the latest version and printable form of this document.

What You Should Know

Although not required, it is recommended that you follow *Getting Started with Sun ONE Application Server 7* on the Sun Microsystems documentation web site prior to exercising this tutorial. *Getting Started with Sun ONE Application Server 7* introduces you to the application server's feature set, architecture and basic administrative operations.

How This Guide is Organized

This guide is organized as follows:

- Introducing Sun ONE Studio 4, Enterprise Edition for Java
- Setting Up Your Development Environment
- Developing Session Beans and Web Applications
- Summary

Documentation Conventions

File and directory paths are given in Windows format (with backslashes separating directory names). For Unix versions, the directory paths are the same, except forward slashes are used instead of backslashes to separate directories.

This guide uses URLs of the form: `http://server.domain/path/file.html`, where:

- server is the name of the server where you are running the application.
- domain is your internet domain name.
- path is the directory structure on the server.
- file is an individual filename.

The following table shows the typographic conventions used throughout Sun ONE documentation

Typographic Conventions

Typeface	**Meaning**	**Examples**
`Monospaced`	The names of files, directories, sample code, and code listings; and HTML tags	Open `Hello.html` file. `<HEAD1>` creates a top level heading.
Italics	Book titles, variables, other code placeholders, words to be emphasized, and words used in the literal sense	See Chapter 2 of the *Migrating and Redeploying Server Applications Guide*. Enter your *UserID*. Enter *Login* in the Name field.
Bold	First appearance of a glossary term in the text	**Templates** are page outlines.

Chapter 1

Introducing Sun ONE Studio 4, Enterprise Edition for Java

Sun ONE Studio 4, Enterprise Edition for Java is an integrated development environment (IDE) for Java technology developers. Based on the modular, open source NetBeans[tm] Tools Platform, the Sun ONE Studio IDE is ideal for building and deploying Web services across multiple hardware and software platforms. This modular, extensible IDE enables fast adoption of new technologies from Sun, Sun's partners, and the community.

In addition to the J2EE[tm] web application, XML and other development features offered in Sun ONE Studio 4, Community Edition, the Sun ONE Studio 4, Enterprise Edition for Java product offers the following features and benefits:

Studio Feature	**Benefit to Developers**
Enterprise JavaBeans[tm] (EJB[tm]) 2.0 Workshop	Rapid development of Java 2 Platform, Enterprise Edition (J2EE) 1.3 Enterprise Applications. Create, extend, and test EJB components such as message-driven, session, BMP entity, and CMP entity beans. Work can be done in a logical bean view that allows you to focus on business methods while Sun ONE Studio handles the plumbing.

Studio Feature	Benefit to Developers
Assemble applications from EJBs and package applications for deployment	Rapid deployment of enterprise applications while minimizing errors. Create, develop, and deploy J2EE applications and export J2EE standard EAR files. The assembly module provides: • Automatic generation of EJB jar and EAR files • Property sheets for editing deployment descriptors • Automatic recognition of standard EJB jar files • Wizard-driven EJB module and J2EE application assembly • Resolve/Link EJB references • Resolve application level security roles • Override or propagate EJB deployment descriptor information • Explorer view of application • Create and develop EJB modules and export EJB .jar file.
Application server integration for deployment	Shorter iterative development cycles through integrated tools.

For more product information, see the Sun ONE Studio 4 page on the Sun Microsystems web site.

Sun ONE Studio 4 Integration with Sun ONE Application Server 7

When using Sun ONE Studio 4, Enterprise Edition for Java with Sun ONE Application Server 7, enterprise developers benefit from many features that combine to give seamless integration between the IDE and the application server.

Studio Feature	Sun ONE Application Server Extension
CMP Mapping	Developers can browse database tables, select related tables and automatically generate Container Managed Persistence (CMP) EJBs.

Studio Feature	Sun ONE Application Server Extension
Server Runtime Control	Developers can easily register both local and remote application servers and start and stop application server instances.
Resource Configuration	Before deploying applications, developers can register J2EE resources in any of the registered application servers. JDBC[tm] resources and connection pools, JMS resources, and a variety of other resources can be configured from within the IDE.
Application Deployment	Developers can select from the list of registered application servers and leverage the dynamic ("hot") deployment and redeployment features supported in Sun ONE Application Server 7.
Debugging and Log Viewing	Debugging against deployed applications on both local and remote application server instances is simple. No manual configuration of the application servers is necessary. While debugging applications, developers can also view the server event log files from within the IDE.

Proceed to Setting Up Your Development Environment to either install the Sun ONE Studio 4 IDE or to configure an existing installation of the IDE to work the application server.

Chapter 2

Setting Up Your Development Environment

Before configuring your development environment, take a few moments to learn about how the Sun ONE Studio 4, Enterprise Edition for Java and Application Server 7 products are integrated.

Sun ONE Studio 4 and Application Server Integration

The application server software that extends the IDE's functionality is generally referred to as the application server's "plugin" for the IDE. The plugin consists of a series of JAR and configuration files that are installed in either the IDE's installation directory or a user's IDE directory.

Installing the IDE Plugin

There are three means by which a user can install the application server plugin in a Studio 4 installation:

1. Plugin is installed as part of a combined application server and IDE installation.
2. Plugin is installed as a component using the application server's installation program.

3. Plugin is downloaded and installed as a NetBeans[tm] module from the Sun ONE Studio Update Center.

Although editing of Sun ONE deployment descriptors and development of CMP EJBs can be accomplished by using only the plugin and the IDE, the most common scenario is to also configure the IDE connect to an application server deployment environment for the purpose of deploying and testing applications.

Connecting the IDE to Application Server Deployments

The plugin relies on a small number of administrative client JAR files that enable the plugin to communicate with the administrative server of an application server deployment. Once the plugin is installed in either the IDE or a user's IDE directory and the administrative client libraries have been made available on the same machine, a property must be set to inform the plugin about the location of the administrative client libraries.

Once the administrative client libraries are installed and the IDE is configured properly, the developer is able to register any number of either local or remote application server deployment environments in the IDE. Any one of these registered application server deployments can be designated as the default server in the IDE.

When you install the evaluation distribution of Sun ONE Application Server 7 that also includes Sun ONE Studio 4, Enterprise Edition for Java, the plugin and dependent administrative client files are installed automatically. In the case of a combined installation, the plugin is automatically configured to point to the application server's installation location so that the plugin can access the administrative client libraries.

Setting Up Your Development Environment

Your first step in working with Sun ONE Studio 4 and Application Server 7 is to set up the two environments to work together. Based on the products that you've installed on your system(s), use the following table to determine the steps required to set up your development environment.

NOTE **Application Server Must be Installed**: The following starting points assume that you've already installed the application server. If you have not already installed the application server, download Sun ONE Application Server 7 and install it according to the instructions contained in the *Getting Started with Sun ONE Application Server 7* guide on the Sun Microsystems documentation web site. If you have not already installed Sun ONE Studio 4, Enterprise Edition for Java, download the application server distribution containing the IDE. Installation of the combined Sun ONE Application Server 7 and Sun ONE Studio 4, Enterprise Edition for Java distribution results in a minimal number of post installation configuration steps.

CAUTION **Solaris 9 Users**: If the Sun ONE Application Server 7 product was installed as part of a Solaris 9 installation, then you must follow the *Getting Started with Sun ONE Application Server 7* guide on the Sun Microsystems documentation web site in order to set up your own application server and PointBase database environment prior to following this tutorial. If you do not follow the set up instructions in the Getting Started guide, you will not be able to complete this tutorial. If you find the directory `/usr/appserver/` in your Solaris 9 installation, then the application server was installed as part of the Solaris 9 installation. If this is the case, then you need to first follow the set up instructions for the application server and PointBase database as described in the Getting Started guide.

Starting Point	Description	Steps
Application Server and Studio 4 Installed Together	Evaluation distribution of Sun ONE Application Server 7 including Studio 4 has already been installed on your system.	Register Target Application Server
Studio 4 Not Yet Installed or Studio 4 Installed Separately from the Application Server	Products installed separately.	1. Install Sun ONE Studio 4, Enterprise Edition for Java (if not already installed) 2. Install IDE Plugin (Sun ONE Application Server 7 Plugin for Studio 4) 3. Register Target Application Server

Studio 4 and Application Server Installed on Separate Machines	Remote development server scenario.	1. Install IDE Plugin (Application Server 7 Plugin for Studio 4 Enterprise Edition) on Development Workstation. 2. Register Target Application Server

Install Sun ONE Studio 4, Enterprise Edition for Java

If you have not already installed the IDE, download Sun ONE Studio 4, Enterprise Edition for Java and follow the installation instructions in the *Getting Started Guide for Sun ONE Studio 4, Enterprise Edition for Java*. The *Getting Started Guide for Sun ONE Studio 4, Enterprise Edition for Java* can be accessed through either the Documentation section of the Sun ONE Studio 4 product page or docs.sun.com.

When installing the IDE, it is recommended that you use either Java[tm] 2 SDK (J2SDK), Standard Edition 1.4.0_02 or greater so as to match the J2SDK requirements of Sun ONE Application Server 7. Refer to the *Sun ONE Studio 4, Enterprise Edition for Java Getting Started Guide* and *Sun ONE Application Server 7 Installation Guide* for detailed J2SDK compatibility information.

Once you've installed the IDE according to the instructions in the IDE's Getting Started Guide, continue following instructions in this section to install the Sun ONE Application Server 7 plugin and to register a Sun ONE Application Server 7 runtime environment as the default application server in the IDE.

Install IDE Plugin

If you installed the combined Sun ONE Application Server 7 and Sun ONE Studio 4, Enterprise Edition for Java evaluation distribution, then you do not need to install the IDE plugin. In this case, proceed to the section Register Target Application Serverto complete the IDE set up process.

If you did not install the combined application server and IDE distribution, follow these instructions to install the application server's plugin for the IDE. The plugin installation instructions vary depending on whether the IDE is installed on the same machine as the application server or the IDE and application server are installed on different machines.

- Application Server and IDE Installed on Same Machine
- Application Server Installed on Remote Development Server

Application Server and IDE Installed on Same Machine

If the application server is installed on the same machine as the IDE but you did not install the combined application server and IDE products, you must install the application server's plugin for the IDE. There are two ways in which you can accomplish the plugin installation:

- Download Plugin from the Sun ONE Studio Update Center
- Install the Plugin Using the Sun ONE Application Server 7 Installer

Download Plugin from the Sun ONE Studio Update Center

The application server's plugin for the IDE is available as a NetBeans Module file (`.nbm` file) through the Sun ONE Studio Update Center. You can use the IDE's *Update Center Wizard* to connect to the update center and install the Sun ONE Application Server 7 IDE plugin module.

NOTE **Installing Manually Downloaded Plugin Module**: If you obtained the `.nbm` file for the IDE plugin through some means other than the Sun ONE Studio Update Center, you can use the IDE's *Update Center Wizard* to install the `.nbm` file directly. On the first screen of the *Update Center Wizard* select the *Install Manually Downloaded Modules* option to install the plugin module directly.

1. In the IDE, select *Tools -> Update Center...* to access the Sun ONE Studio Update Center.

   The *Select Modules to Install* window is displayed.

2. Under *Available Updates and New Modules*, select the Sun ONE Application Server 7 plugin module and move it to the *Include in Install* portion of the window. Click *Finish* to begin installation of the plugin module.

To complete the installation of the plugin module, the IDE must be restarted.

NOTE **Location of Installed Modules**: When you install a module via the Update Center facility of the IDE, the files that make up the module are installed in your user's IDE directory rather than in the installation directory of the IDE. This approach ensures that modules installed by users through the update center do not conflict with one another.

Proceed to the section Register Target Application Server to complete the IDE set up process.

Install the Plugin Using the Sun ONE Application Server 7 Installer

If you have access to the non-evaluation distribution of Sun ONE Application Server 7, you can use the application server's installation program to install the application server's plugin.

1. Start the application server's installation program.
2. In the *Select Installation Directory* screen, ensure that the location of your existing application server installation directory is specified.
3. In the components selection screen, select the component named *Support for Sun ONE Studio 4, Enterprise Edition for Java*.
4. When prompted for the installation location of the IDE, enter the appropriate location and continue with the installation process.

If the IDE is running, then restart it and proceed to the section Register Target Application Server to complete the IDE set up process.

Application Server Installed on Remote Development Server

If the application server is not installed on the same machine as the IDE and you plan to use the IDE to work with an application server installed on a remote server, then you need to install both the application server's plugin for the IDE and the application server's administrative client support on the development workstation.

If the application server is already installed on the same machine as the IDE and you'd like to use the IDE to work against a remote application server, the application server's administrative client support is already installed. In this case, proceed to the section Register Target Application Server to complete the IDE set up process.

To install the application server's plugin for the IDE and the administrative client support, you can use the non-evaluation distribution of the Sun ONE Application Server 7 product. In this distribution the installation program enables you to select a component named *Support for Sun ONE Studio 4, Enterprise Edition for Java*. Selection of this component results in the installation of both the application server's plugin for the IDE and the administrative client support.

Alternatively, if you have access to only the evaluation distribution of the application server, you can install the complete application server on the development workstation and follow the steps in Application Server and IDE Installed on Same Machine to install the application server's plugin for the IDE.

In any case, once you've installed the plugin and the administrative client support, proceed to the next section to complete the IDE set up process.

Register Target Application Server

Once you've installed Sun ONE Studio 4, Enterprise Edition for Java and the application server's plugin, you are ready to register a target Sun ONE Application Server 7 runtime with the IDE.

1. Start the Application Server
2. Start the IDE
3. Set Sun ONE Application Server as the Default Server

Start the Application Server

If the application server is not already started on either your local workstation or on a development server, start the application server at this time.

The application server must be started in order for you to register an application server with the IDE.

Start the Application Server on Windows Platforms

To start the application server from the Windows desktop, select:

Start -> Programs -> Sun Microsystems -> Sun ONE Application Server 7 -> Start Application Server

After selecting *Start Application Server*, a command window is displayed to show the status of the startup process.

A second command window also appears on your desktop. This command window is a read only representation of the server event log file content. The second command window remains on your desktop as long as the associated application server instance is running. As the initially configured server starts, event messages appear in the second command window. After a few seconds, a message confirming that the server instance has started successfully appears in the command window.

Start the Application Server on UNIX Platforms

On UNIX platforms, execute the following command:

```
$ <appserver_install_dir>/bin/asadmin start-domain --domain
domain1
```

Execution of this command starts the initially configured domain, named “domain1”, on your system.

Start the IDE

Start the IDE by running the program executable, as described in this section. For more detailed information about starting the IDE, see the Getting Started Guide for the Sun ONE Studio 4, Enterprise Edition on the Sun Microsystems web site.

The first time you use the IDE, several dialog windows appear that allow you to do the following:

- Set the user directory to store user and IDE configuration information.
- Import settings from a previous installation of the IDE.
- Configure a proxy server if you are behind a firewall.
- Set your preferred web browser.
- Register the product. After registering, you will be eligible for any product updates that are released.

NOTE **Registering the Product:** It is highly recommended that you register your product installation so that you can access the NetBeans and Sun ONE Studio update centers. To register for updates, you need the serial number of your IDE installation. The serial number is displayed in the Registration Wizard. This wizard can be displayed by accessing *Help* -> *Registration Wizard* in the IDE.

The update centers can be accessed via the *Tools* -> *Update Center* item in the IDE.

Start the IDE on Windows Platforms

There are three methods of starting the IDE:

1. From the Windows desktop, select:

   Start -> *Programs* -> *Sun ONE Studio 4u1 EEfJ* -> *Sun ONE Studio*

2. Double-click the *Sun ONE Studio 4u1 EEfJ* icon on your desktop.

3. Run the `runide.exe` executable from the command line. For example:

   ```
   c:\> <ide_install_dir>\bin
   ```

Start the IDE on UNIX Platforms

A `runide.sh` script is located in the *ide_install_dir*`/bin/` directory. Start the IDE by executing following command in a terminal window:

```
<ide_install_dir>/bin/runide.sh
```

Set Sun ONE Application Server as the Default Server

Since the exercises in the guide depend on the Sun ONE Application Server, you need to make sure that an available Sun ONE Application Server is registered and is set as the default server in the IDE.

Once you set an application server as a default server, all deployment, execution and debugging operations selected from within the IDE automatically targets the default server.

Ensure Plugin is Configured with Application Server Installation Directory

In order for the IDE and plugin to interact with a Sun ONE Application Server 7 runtime, the plugin must be configured to point to the installation location of either the fully installed application server or the application server's administrative client libraries. If you installed the evaluation distribution of Sun ONE Application Server 7 that also included the IDE, the plugin has already been configured with the appropriate application server installation directory.

1. In the IDE, bring up the *Explorer* window and *Runtime* tab at the bottom of this window. If you do not see the *Explorer* window, simply select *View->Explorer* from the main IDE window.
2. Expand the *Server Registry*, *Installed Servers*, and *Sun ONE Application Server 7* nodes.

CAUTION **Sun ONE Application Server 7 Does Not Exist**: If you do not see *Sun ONE Application Server 7* node in the *Installed Servers* area, then it is likely that the plugin module for the application server has not yet been installed. See the Install IDE Plugin section for instructions on installing the application server plugin.

---

3. Right-click the *Sun ONE Application Server 7* node and select *Properties*.

   The properties editor is displayed.

4. Ensure that the *Sun ONE Application Server Home* property is configured with the location of either the application server or application server's administrative client component.

   For example, on Windows systems, this property value may appears as follows:

   ```
   c:\Sun\AppServer7
   ```

   On Solaris[tm] systems, examples of the property setting include:

   ```
   /usr/appserver
   /opt/SUNWappserver7
   <home_dir>/appserver7
   ```

Register a Sun ONE Application Server

Now that you've ensured that the plugin is configured properly, your next step is to register a specific application server runtime environment with the IDE.

CAUTION **Application Server Must Be Running Prior to Registration**: You must start the application server prior to attempting to register it in the IDE. Specifically, the administrative server must be running prior to registration.

To register an application server:

1. Under *Installed Servers* section, expand the *Sun ONE Application Server 7* node.

   If you see a *localhost:port* node under the *Sun ONE Application Server 7* node, an application server instance is already registered. If the named server is the one with which you plan to work, you can proceed to the Verify that the Application Server Instance is Running section.

2. If you either do not see a registered application server or the registered server is the not the server of interest, right-click the *Sun ONE Application Server 7* node and choose *Add Admin Server*.

   The *Add Admin Server* dialog box appears.

3. Type `localhost` in the *Admin Server Host* field.

NOTE **Accessing a Remote Application Server**: You can also register an application server running on a remote machine. To register the application server on an another machine, type the remote machine's hostname or IP address instead of `localhost` in the *Admin Server Host* field.

4. Type your server’s administrative server port number in the *Admin Server Port* field.

CAUTION **Forgot the Admin Server Port Number?** If you do not remember the HTTP server port number of the administrative server, you can inspect the administrative server's configuration file to determine the HTTP server port number.

1. Navigate to the following directory:

```
<appserver_domain_config_dir>/domain1/admin-server/conf/
```

2. Open the `server.xml` file in your favorite editor. Look for the element:

```
<http-listener id="http-listener-1" address="0.0.0.0"
port="4848"...
```

In this case, port 4848 is the HTTP port number in use.

5. Type the administrative user name in the *User Name* field.
6. Type the administrative password you specified during installation in the *User Password* field.

CAUTION **Forgot the Username or Password?** If you do not remember the administrative user name that was supplied during installation, try the user name "admin". This is the default user name specified in the server configuration dialog during installation. If you still cannot determine the user name, look in the following file:

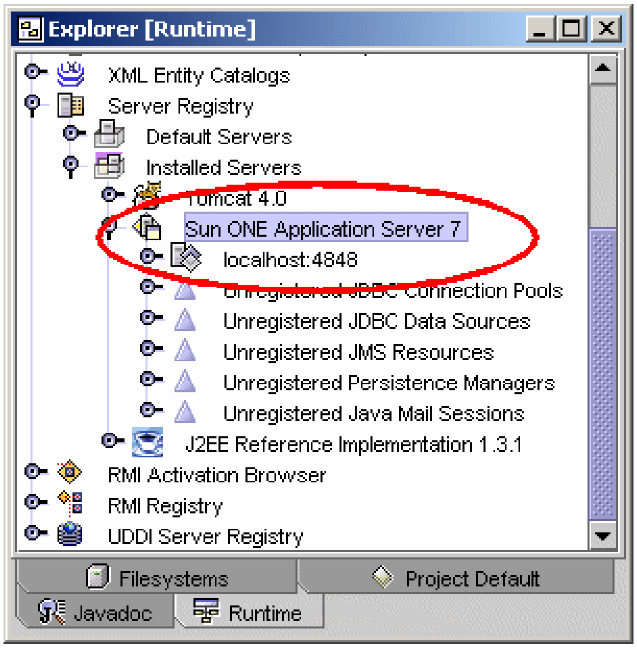
```
<appserver_domain_config_dir>/domain1/admin-server/confi
g/admpw
```

This file contains the administrator's user name followed by the encrypted form of the administrative user's password.

If you do not remember the administrator's password, then you can delete the administrative domain using the `delete-domain` subcommand of `asadmin` and create a new domain with a new administrative password.

7. Click OK.

   You should now be able to see the application server listed as an installed server.

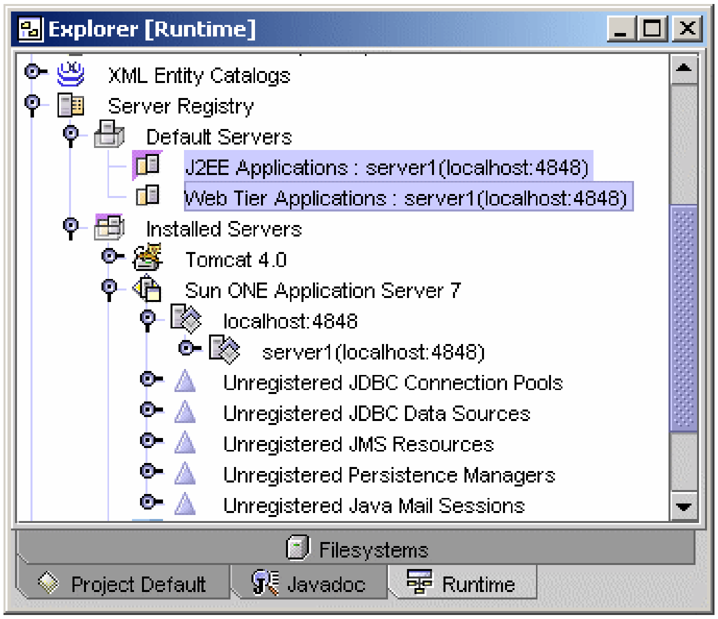

Explorer [Runtime]
XML Entity Catalogs
Server Registry
Default Servers
Installed Servers
Tomcat 4.0
Sun ONE Application Server 7
localhost:4848
Unregistered JDBC Connection Pools
Unregistered JDBC Data Sources
Unregistered JMS Resources
Unregistered Persistence Managers
Unregistered Java Mail Sessions
J2EE Reference Implementation 1.3.1
RMI Activation Browser
RMI Registry
UDDI Server Registry
Filesystems
Project Default
Javadoc
Runtime

8. Expand the `localhost:`*port* administration server node, you can see a node named *server1(localhost:4848)* that corresponds to the application server instance that was created during installation of the application server. You will be using this instance throughout this tutorial.

9. Right-click the *server1(localhost:4848)* node and select *Set As Default*.

10. Verify that the default server has been set correctly by expanding the *Default Servers* node under the *Server Registry* area.

You should see the *server1(localhost:4848)* item as the default server instance.


Explorer [Runtime]
XML Entity Catalogs
Server Registry
Default Servers
J2EE Applications : server1(localhost:4848)
Web Tier Applications : server1(localhost:4848)
Installed Servers
Tomcat 4.0
Sun ONE Application Server 7
localhost:4848
server1(localhost:4848)
Unregistered JDBC Connection Pools
Unregistered JDBC Data Sources
Unregistered JMS Resources
Unregistered Persistence Managers
Unregistered Java Mail Sessions
Filesystems
Project Default
Javadoc
Runtime

Verify that the Application Server Instance is Running

In this section, you will examine whether or not the application server instance is running and to start it if it is not running.

1. Right-click the `server1(localhost:`*port*) instance node and select *Status*.

   The *Server Status* dialog box appears. If the *Start Server Instance* button is disabled, the application server instance is already running and you can skip the next step, number 2.

2. Click the *Start Server Instance* button.

   After several seconds, the application server instance starts, and the *Server Status* dialog box indicates that the instance is started.

3. Click *OK* to close the *Server Status* dialog box.

   Now that the application server is started and the IDE is configured properly, you are ready to proceed to Developing Session Beans and Web Applications.

Chapter 3

Developing Session Beans and Web Applications

In this exercise you will create a simple stateless session bean and a web front end using the IDE. After testing the session bean, you will compose a J2EE[tm] application, verify that the application works properly, modify several components in the application and then step through various parts of the application using the IDE's debugging facilities.

- Becoming Familiar with the Sample
- Creating a Session Bean with the EJB[tm] Builder
- Testing the Session Bean
- Packaging the Enterprise Bean in an EJB[tm] Module
- Creating the Web Application
- Assembling the J2EE[tm] Application
- Setting Up Database Connectivity
- Deploying and Running the J2EE[tm] Application
- Modifying the Application
- Debugging the Application

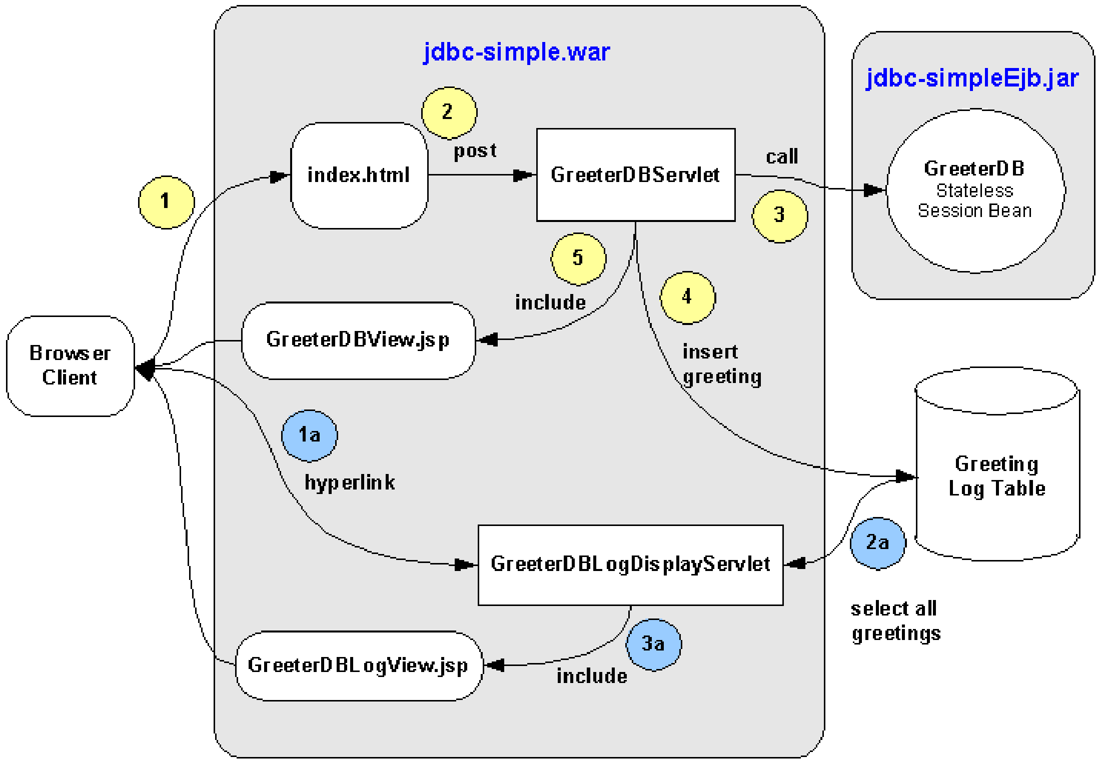
Proceed to Becoming Familiar with the Sample for a brief introduction to the sample application prior to deploying and exercising it.

Becoming Familiar with the Sample

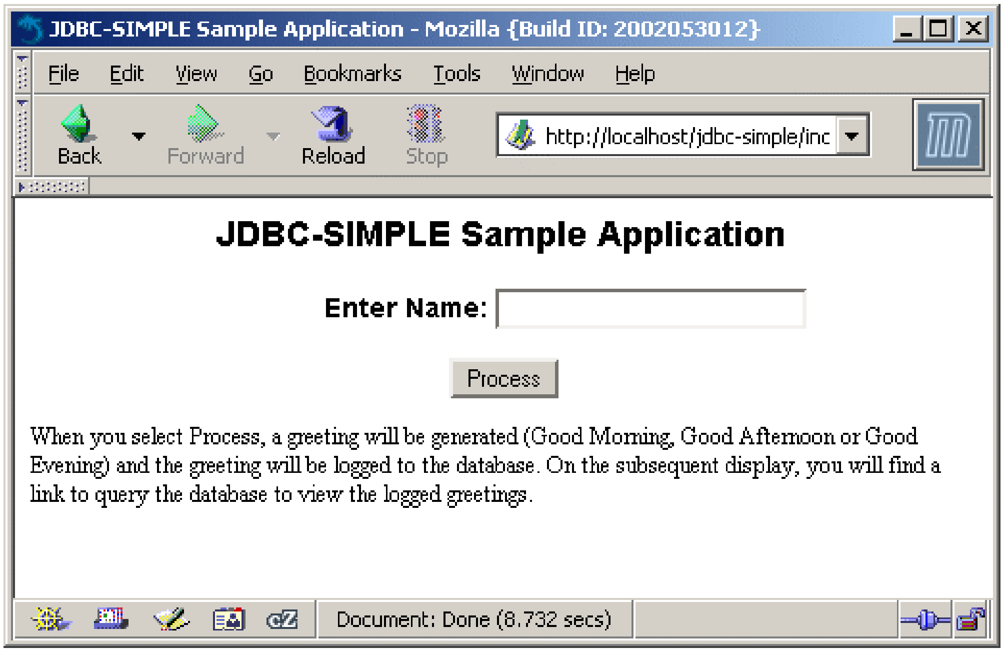
If you are already familiar with the JDBC[tm] Simple sample application (it is the sample used in *Getting Started with Sun ONE Application Server 7)*, proceed to the next section, Creating a Session Bean with the EJB[tm] Builder.

The sample application that you will be using during this exercise consists of a web application module and an EJB module containing a single stateless session bean. The web and EJB modules are packaged in an Enterprise Application Archive (EAR) file. Although patterned after basic "Hello World" applications, this sample displays a greeting based on the user's input and the time of day. Since each greeting is recorded to a database table, the user also has the option to list all previously generated greetings.

The display greeting path through the application is represented by the yellow numbered steps in the following diagram while the greeting log display path through the application is represented by the lettered blue steps in the diagram.



When the user accesses the application, the first page of the application prompts the user to enter a name that will be displayed in a subsequent greeting page.


JDBC-SIMPLE Sample Application - Mozilla {Build ID: 2002053012}
File
Edit
View
Go
Bookmarks
Tools
Window
Help
Back
Forward
Reload
Stop
http://localhost/jdbc-simple/inc
Process
Document: Done (8.732 secs)

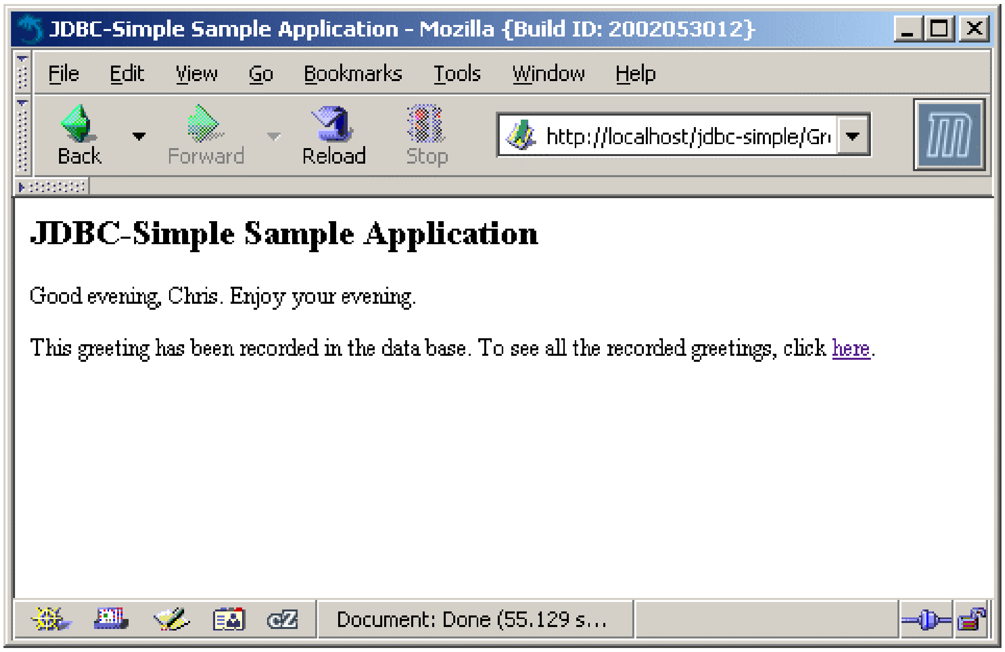
JDBC-SIMPLE Sample Application

Enter Name:

When you select Process, a greeting will be generated (Good Morning, Good Afternoon or Good Evening) and the greeting will be logged to the database. On the subsequent display, you will find a link to query the database to view the logged greetings.

After entering a name and clicking on the *Process* button, the following page is displayed with the appropriate greeting for the time of day.

In the background, the GreeterDBServlet accessed the stateless session bean to determine the appropriate greeting ("morning", "afternoon" or "evening"). The GreeterDBServlet sets the appropriate greeting message in the request object of the servlet request and specifies a JavaServer Pages[tm] page to display the result.


JDBC-Simple Sample Application - Mozilla {Build ID: 2002053012}
File Edit View Go Bookmarks Tools Window Help
Back Forward Reload Stop
http://localhost/jdbc-simple/Gr

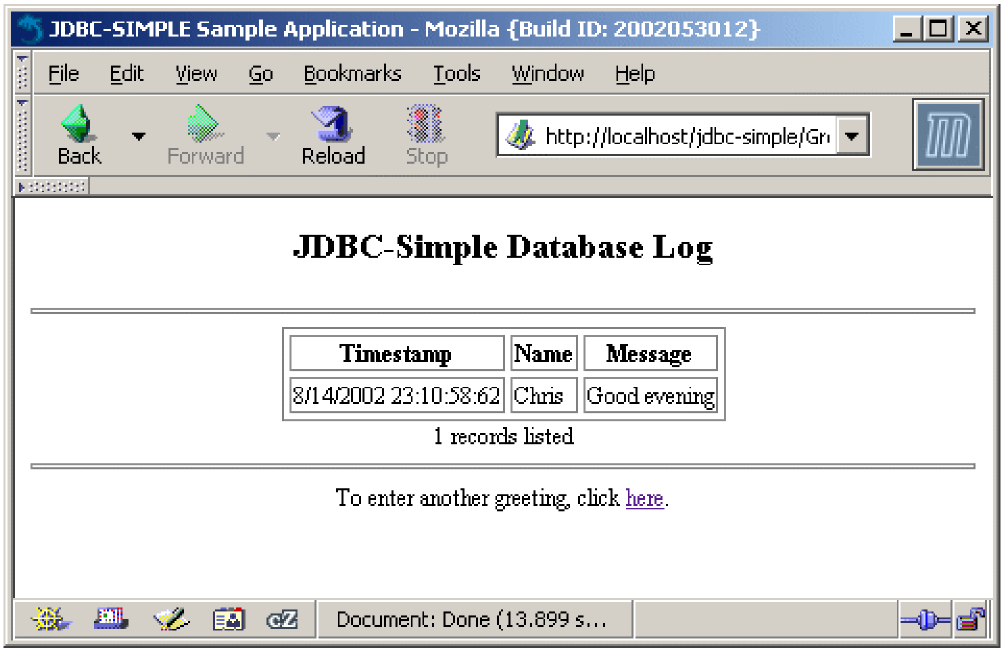
JDBC-Simple Sample Application

Good evening, Chris. Enjoy your evening.

This greeting has been recorded in the data base. To see all the recorded greetings, click here.

Document: Done (55.129 s...

The greeting is displayed. When the user elects to list all previously generated greetings, the GreeterDBLogDisplayServlet performs a select all on the database table of greetings and specifies a JSP[tm] page to display the result


JDBC-SIMPLE Sample Application - Mozilla {Build ID: 2002053012}
File Edit View Go Bookmarks Tools Window Help
Back
Forward
Reload
Stop
http://localhost/jdbc-simple/Gr

JDBC-Simple Database Log

Timestamp	Name	Message
8/14/2002 23:10:58:62	Chris	Good evening

1 records listed

To enter another greeting, click here.

Document: Done (13.899 s...

Proceed to Creating a Session Bean with the EJB[tm] Builder to create the stateless session bean from scratch.

Creating a Session Bean with the EJB[tm] Builder

The session bean that you will create is called `GreeterDB`. It generates a greeting message based on the current time upon request.

1. Create the Session Bean
2. Modify JNDI Name
3. Modify Automatically Generated Methods
4. Create a Business Method

Create the Session Bean

To create the session bean using the IDE's EJB Builder facility:

1. In the IDE, click the Explorer window's *Filesystems* tab.
2. Select the *Filesystems* node, right-click, select *Mount* -> *Local Directory*.

   The *New Wizard - Local Directory* window is displayed.
3. Navigate to the IDE user directory that you specified during the first startup of the IDE:

   `<ide_user_dir>/sampledir/examples`

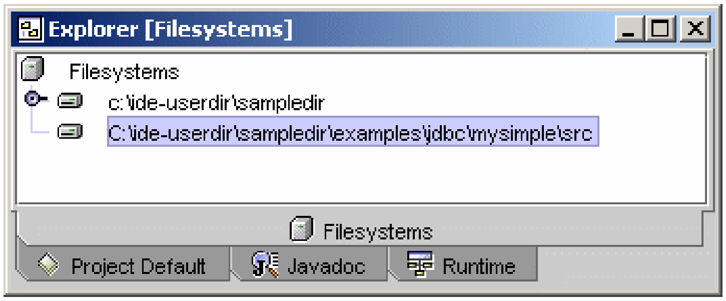
   For example:

   `C:\ide-userdir\sampledir\examples`

   The user directory is automatically created when you start the IDE for the first time. You may have specified a different base user directory than presented in this example, but the `sampledir` directory is created automatically.
4. Click the *Create New Folder* icon in the upper left hand portion of the window.

   A folder named `New Folder` appears.
5. Click on the `New Folder` item and enter the name `"jdbc"`.
6. Double click the newly named `"jdbc"` directory and create a new folder named `"mysimple"` under this directory.
7. Repeat this operation to create a directory named `"src"` under the `mysimple` directory.
8. Click once on the `"src"` directory and click the *Finish* button.

   The newly created `jdbc/mysimple/src` directory appears as a mounted filesystem in the Explorer window.



9. Right-click the new filesystem and choose *New -> Java Package*.

   You will use this Java package to hold the EJB source code of the application.

10. Name the new Java package:

    `samples.jdbc.simple.ejb`

    Click *Finish*.

    A new directory named samples`/jdbc/simple/ejb` appears under the `mysimple/src` filesystem.

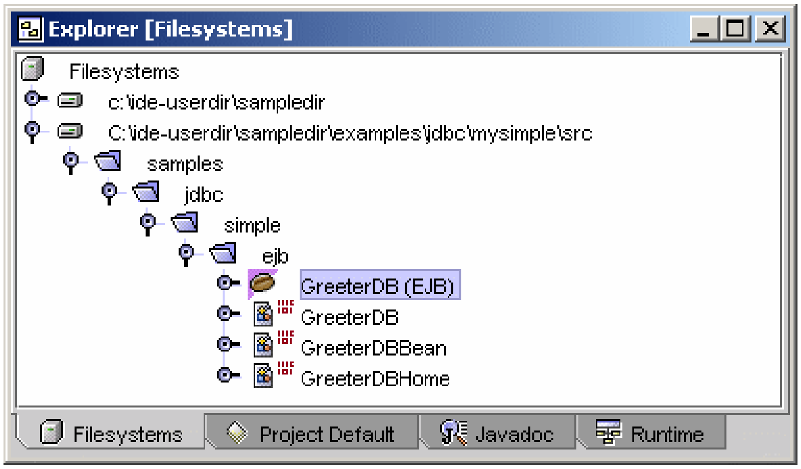
11. Right-click the ejb directory node and choose *New -> J2EE -> Session EJB*.

    The *Session Bean Name* and *Properties pane* of the *New Wizard* is displayed.

12. Type `GreeterDB` in the *EJB Name* field and click *Finish*.

    The default settings are used when creating the session bean and then the `GreeterDB` session bean is displayed in the Explorer window.

    Note the presence of a new *GreeterDB (EJB)* node and three other nodes:



Explorer [Filesystems]

```
Filesystems
  c:\ide-userdir\sampledir
  C:\ide-userdir\sampledir\examples\jdbc\mysimple\src
    samples
      jdbc
        simple
          ejb
            GreeterDB (EJB)
            GreeterDB
            GreeterDBBean
            GreeterDBHome
```

The *GreeterDB (EJB)* node is a logical representation of the session bean while the three other nodes represent the class files of the remote, implementation and home interfaces. The IDE creates the minimal set of required methods in the EJB implementation class file.

Modify JNDI Name

To avoid the possibility of naming conflicts with previously deployed versions of the JDBC Simple application, you need to modify the JNDI Name of the EJB.

1. Right-click the `GreeterDB (EJB)` node and select *Properties*.
2. Select the *Sun ONE AS* tab.
3. Enter `ejb/my-jdbc-simple` in the *JNDI Name* field.
4. Close the *Properties* window.

This JNDI name will uniquely identify this EJB throughout the application server instance to which the EJB is deployed.

Modify Automatically Generated Methods

Your next step is to enhance the automatically generated methods of the session bean. These modifications will result in output being written to the stdout stream when each standard method is invoked by the EJB container of the application server. Since output written to stdout is automatically redirected to the application server's event log, you will see this output in the server's log file when you test the session bean.

To update methods:

1. Double-click the *GreeterDBBean* node that represents the implementation class of the session bean.

   The source editor shows the `GreeterDBBean.java` file.

2. Add the following code (the bold text only) to the body of the session bean methods to display the status of the bean instance:

```
/**
* @see javax.ejb.SessionBean#ejbActivate()
*/
public void ejbActivate() {
  System.out.println("ejbActivate() on obj " + this);
}
...
/**
```

```
* @see javax.ejb.SessionBean#ejbPassivate()
*/
public void ejbPassivate() {
  System.out.println("ejbPassivate() on obj " + this);
}
...
/**
* @see javax.ejb.SessionBean#ejbRemove()
*/
public void ejbRemove() {
  System.out.println("ejbRemove() on obj " + this);
}
...
/**
* See section 7.10.3 of the EJB 2.0 specification
*/
public void ejbCreate() {
  System.out.println("ejbCreate() on obj " + this);
}
```

3. In the IDE, choose *File -> Save All*.

This will save the *GreeterDB* session bean and the changes that you have made thus far.

NOTE **Why select *Save All*?** In many cases, your work is saved automatically by the IDE. However, use *File -> Save All* whenever you wish to either leave the tutorial for an extended period of time or want to make sure all your work is saved. You may also use the *Save and Save All* buttons in the toolbar save modifications.

Create a Business Method

The EJB Builder automatically generated the required methods of the session bean and you have customized these methods with output statements. Now you need to add a simple business method to the bean. This method, `getGreeting()`, generates a greeting based on the current time.

To create the `getGreeting()` business method:

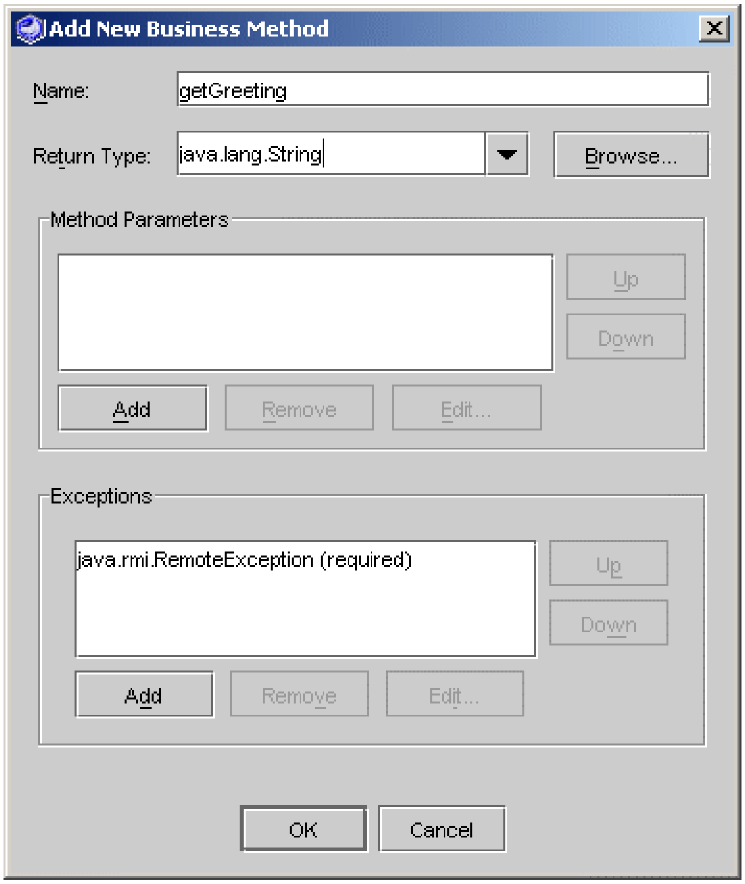
1. Right-click the *GreeterDB (EJB)* node and choose *Add Business Method*.

   The *Add New Business Method* dialog box appears.

2. Type `getGreeting` in the *Name* field.

3. Choose `java.lang.String` in the *Return Type* field.

   The completed dialog window should look like this:



Name: getGreeting

Return Type: java.lang.String

Method Parameters

Exceptions

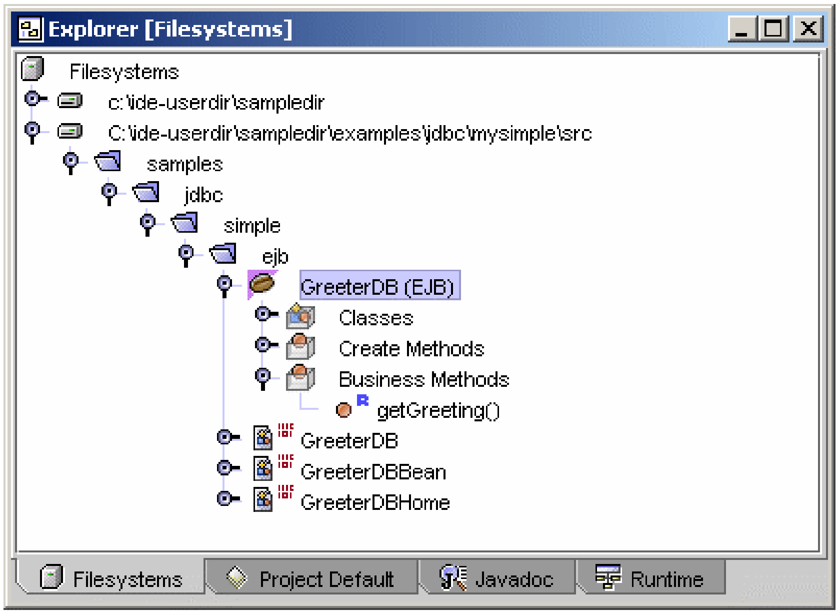
java.rmi.RemoteException (required)

4. Click *OK*.

The `getGreeting()` business method is added to the GreeterDB session bean and its method signature is automatically added to the GreeterDB bean's remote interface.

5. Expand the *GreeterDB (EJB)* node and its *Business Methods* node.

The *getGreeting()* method node is displayed under the *Business Methods* node.



6. Double-click the *getGreeting()* method node.

The source editor displays the content of the bean implementation source file and places the cursor at the start of the `getGreeting()` method.

7. Add the following code (the bold text only) to the body of the `getGreeting()` method:

```
public java.lang.String getGreeting() {
System.out.println("GreeterDB EJB is determining message...");
String message = null;
Calendar calendar = new GregorianCalendar();
int currentHour = calendar.get(Calendar.HOUR_OF_DAY);
if(currentHour < 12) {
    message = "morning";
} else {
    if((currentHour >= 12) &&
            (calendar.get(Calendar.HOUR_OF_DAY) < 18)) {
        message = "afternoon";
        } else {
            message = "evening";
        }
    }
    System.out.println("- Message determined successfully");
    return message;
    }
```

8. Add an `import` statement (the bold text only) for the `java.util.*` package at the top of the source file.

```
package samples.jdbc.simple.ejb;

import javax.ejb.*;
import java.util.*;
...
```

9. In the source editor window, right-click and select *Save* to ensure that your changes have been saved.

10. In the Explorer window, right-click the *GreeterDB (EJB)* node and select *Validate EJB*.

An Output Window specifying "Finished GreeterDB (EJB)" should be displayed.

If this message is not displayed, the code segments above were not copied properly into the source file. Based on the specific error displayed as a result of the validation step, go back and double check your source code.

NOTE **What happens during Validate**? The IDE's EJB Builder contains a custom compiler that not only compiles the EJB source code, but also validates your enterprise beans against the Enterprise JavaBeans[tm] specification.

After the bean validates successfully, close the source code editing window and proceed to Testing the Session Bean to use the IDE's built-in EJB test facility.

Testing the Session Bean

The IDE includes a facility to automatically generate a test environment for the Enterprise JavaBeans[tm]. This feature creates a web-based test client application and packages it along with the enterprise bean. You can use this test application to create an instance of the enterprise bean and to interact with it. In this section you will use this test facility to exercise the GreeterDB bean's `create()` and `getGreeting()` methods.

1. Create the Test Client
2. Deploy the Test Client
3. Run the Test Client

Create the Test Client

When you create a test client, the IDE generates an EJB[tm] module, supporting elements, and a J2EE[tm] application module.

To create a test client for the GreeterDB session bean:

1. Right-click the *GreeterDB (EJB) node* and select *Create New EJB Test Application*.

   The *EJB Test Application* wizard is displayed.

2. Accept all default values and click the *OK* button.

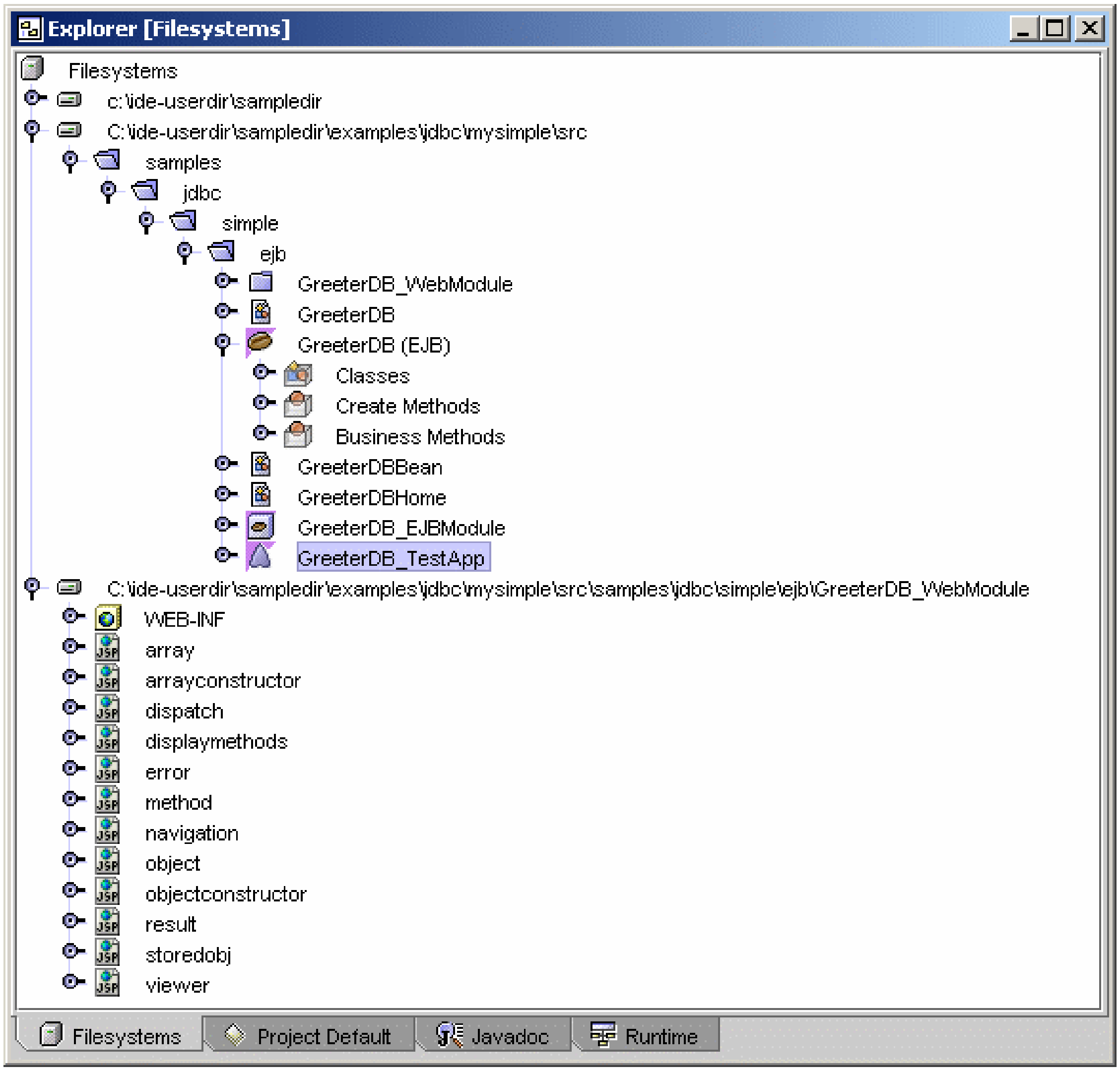
   A progress monitor appears briefly and then goes away when the process is complete.

   Another window is displayed informing you that the web module that was created is also visible in the *Project* tab. It should go away automatically also. If not, click *OK* to close the window.

3. View the resulting test objects in the Explorer.

The IDE has created an EJB module named *GreeterDB_EJBModule*, a web module named *GreeterDB_WebModule*, and a J2EE application named *GreeterDB_TestApp* in the ejb subdirectory.

The EJB module contains the GreeterDB session bean. The web module contains a number of JavaServer Pages[tm] pages and Java class files that make up the test client. The J2EE application includes references to the EJB module and to the web module. In addition, the IDE mounts the web module.



Filesystems
c:\ide-userdir\sampledir
C:\ide-userdir\sampledir\examples\jdbc\mysimple\src
samples
jdbc
simple
ejb
GreeterDB_WebModule
GreeterDB
GreeterDB (EJB)
Classes
Create Methods
Business Methods
GreeterDBBean
GreeterDBHome
GreeterDB_EJBModule
GreeterDB_TestApp
C:\ide-userdir\sampledir\examples\jdbc\mysimple\src\samples\jdbc\simple\ejb\GreeterDB_WebModule
WEB-INF
array
arrayconstructor
dispatch
displaymethods
error
method
navigation
object
objectconstructor
result
storedobj
viewer

NOTE **Confusing list of icons?** To avoid confusion amongst the various nodes representing your business application EJB and the automatically generated test application, you have the option during the creation of the test application to specify both the location of the test application and Java package names used in the test application.

CAUTION **Content of Test Application**: The EJB module, web module, and J2EE application generated by the IDE are not intended to be modified. If you modify them you might not be able to redeploy the J2EE test application.

Deploy the Test Client

Before deploying the test application, ensure that the application server is running and it is set as the default server in the IDE.

To deploy the test application:

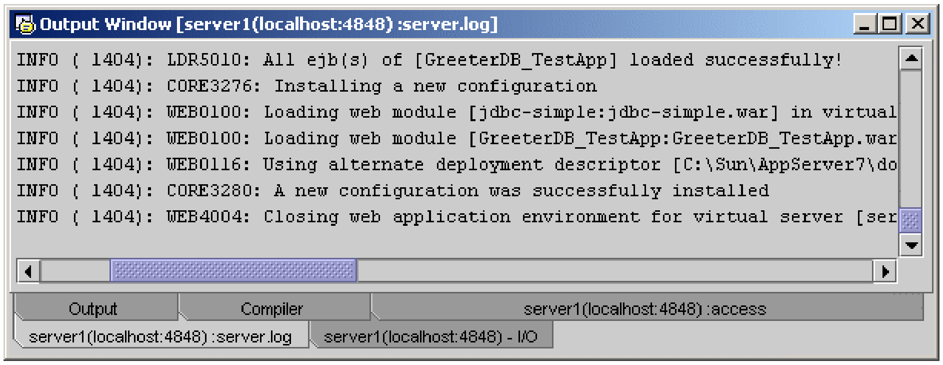
1. Right-click the *GreeterDB_TestApp* J2EE application node and select *Deploy*.

   The progress of the deployment process is displayed in a dialog window.

   See the new tabs in the IDE's Output window that display messages about the deployment.

2. Click the *server1(localhost:port):server.log* tab in the output window.

   The server.log view displays any error conditions that prevent deployment. When the deployment was successful, the output window resembles the following:



Output Window [server1(localhost:4848) :server.log]

```
INFO ( 1404): LDR5010: All ejb(s) of [GreeterDB_TestApp] loaded successfully!
INFO ( 1404): CORE3276: Installing a new configuration
INFO ( 1404): WEB0100: Loading web module [jdbc-simple:jdbc-simple.war] in virtual
INFO ( 1404): WEB0100: Loading web module [GreeterDB_TestApp:GreeterDB_TestApp.war
INFO ( 1404): WEB0116: Using alternate deployment descriptor [C:\Sun\AppServer7\do
INFO ( 1404): CORE3280: A new configuration was successfully installed
INFO ( 1404): WEB4004: Closing web application environment for virtual server [ser
```

Output
Compiler
server1(localhost:4848) :access
server1(localhost:4848) :server.log
server1(localhost:4848) - I/O

3. On the IDE's toolbar, select the *Editing* tab.

   The Explorer window should be displayed. If not, select *View -> Explorer* from the IDE's menu bar.

4. In the Explorer window, select the *Runtime* tab.
5. Expand the *Server Registry*, *Installed Servers*, *Sun ONE Application Server 7*, *localhost:port*, *server1(localhost:port)*, and *Deployed Applications* nodes.

   You should see the *GreeterDB_TestApp* node under the *Deployed Applications* node. If the *GreeterDB_TestApp* node is not listed, right-click on the *Deployed Applications* node and select *Refresh List*.

NOTE **What happens during deployment?** The IDE automatically assembles the EAR file and contacts the administrative server of the specified application server to carry out the deployment.

During the initial deployment of an application containing EJBs, the administrative server automatically generates the EJB's stubs and skeletons. Until the EJB interfaces are changed, subsequent redeployment of the application does not entail stub and skeleton generation. This smart redeployment feature greatly optimizes redeployment operations.

Run the Test Client

Since the test client is web-based, you need to open a web browser window to access the initial test page of the generated web application.

On the initial test client’s web page, you'll invoke the `create()` method of the GreeterDB bean’s home interface to create an instance of the bean. Then you'll execute the `getGreeting()` business method of the bean.

To test the GreeterDB bean:

1. Start a web browser and point it to the following URL:

   `http://localhost:`*port*`/GreeterDB_TestApp`

   Where *port* is the HTTP server port number specified during installation. The default HTTP server port number is 80, but it may be different based on the ports in use during installation.

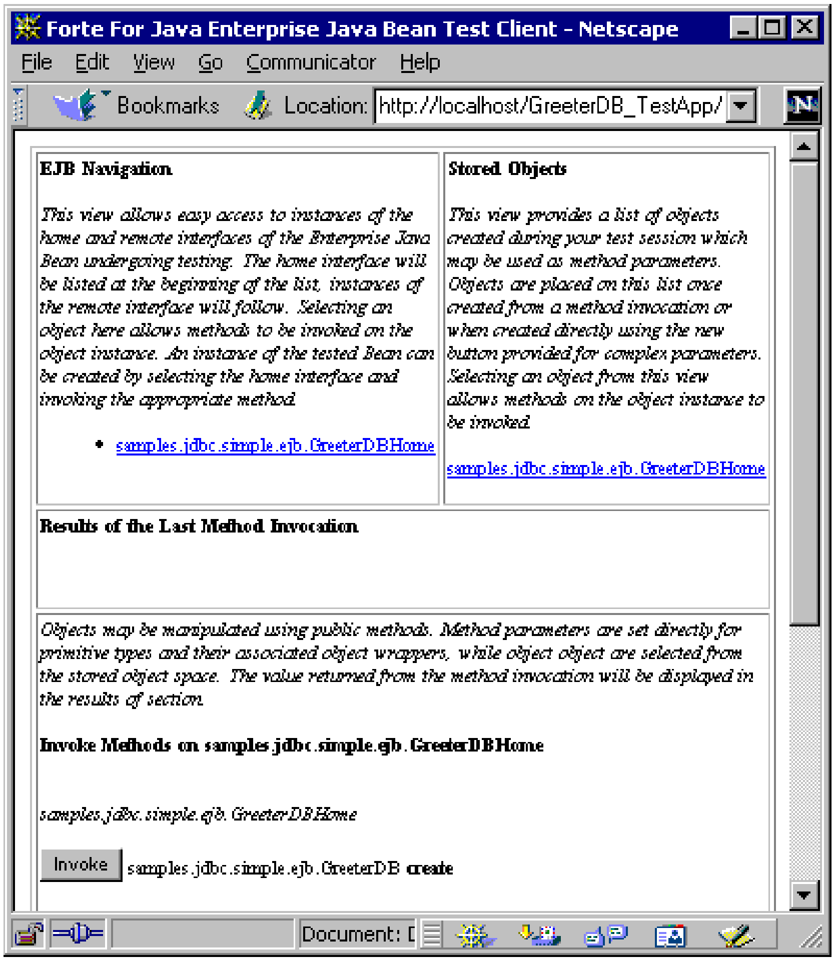
   Your browser displays the test client.

CAUTION **Forgot the HTTP Server's Port Number?** If you do not remember the HTTP server port number of the application server instance, you can determine the appropriate value by looking at the properties of the application server instance.

1. Under the *Runtime* tab of Explorer, expand the *Server Registry*, *Installed Servers*, *Sun ONE Application Server 7*, *localhost:port*
2. Right click the *server1(localhost:port)* object and select *Properties*.
3. Read the HTTP server's port number from the *Server Instance Port* property.

List of instances being tested.

Area where you can enter parameters and invoke methods


Forte For Java Enterprise Java Bean Test Client - Netscape
File Edit View Go Communicator Help
Bookmarks
Location: http://localhost/GreeterDB_TestApp/
Invoke
Document: D

EJB Navigation

This view allows easy access to instances of the home and remote interfaces of the Enterprise Java Bean undergoing testing. The home interface will be listed at the beginning of the list, instances of the remote interface will follow. Selecting an object here allows methods to be invoked on the object instance. An instance of the tested Bean can be created by selecting the home interface and invoking the appropriate method.

- samples.jdbc.simple.ejb.GreeterDBHome

Stored Objects

This view provides a list of objects created during your test session which may be used as method parameters. Objects are placed on this list once created from a method invocation or when created directly using the new button provided for complex parameters. Selecting an object from this view allows methods on the object instance to be invoked.

samples.jdbc.simple.ejb.GreeterDBHome

Results of the Last Method Invocation

Objects may be manipulated using public methods. Method parameters are set directly for primitive types and their associated object wrappers, while object object are selected from the stored object space. The value returned from the method invocation will be displayed in the results of section.

Invoke Methods on samples.jdbc.simple.ejb.GreeterDBHome

samples.jdbc.simple.ejb.GreeterDBHome

samples.jdbc.simple.ejb.GreeterDB **create**

List of objects created during the test session.

Area where results of last method invocation are shown

From this page, you can invoke the methods of the GreeterDBHome remote interface.

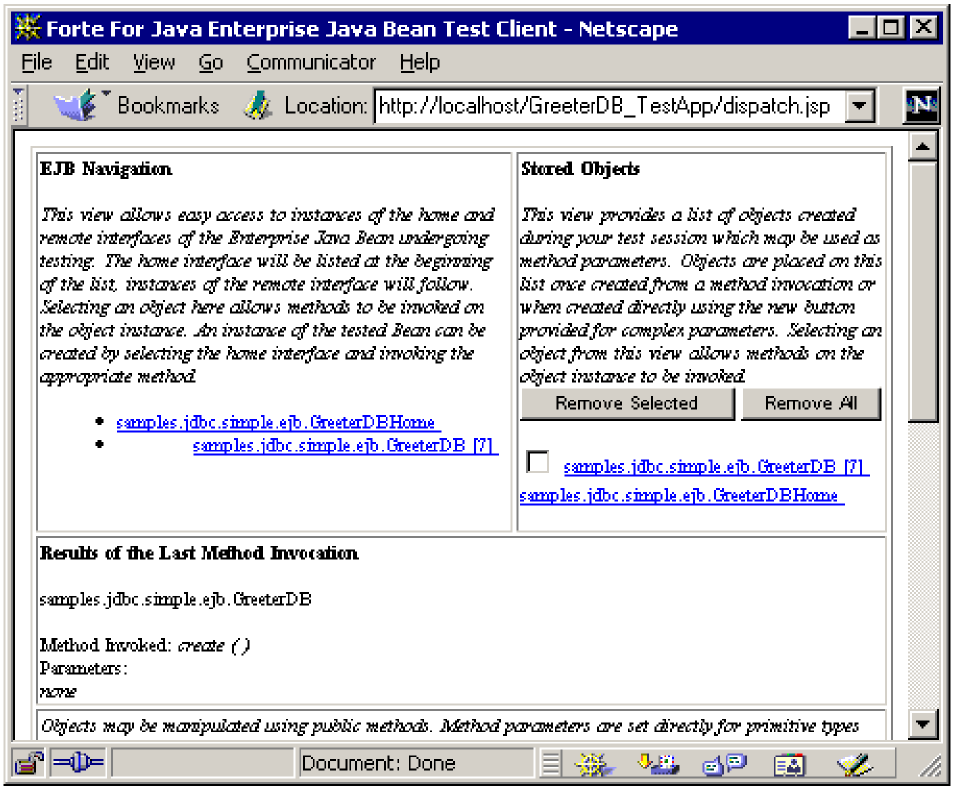
NOTE **Speed of first access**: When you visit a JavaServer Pages[tm] (JSP[tm]) page for the first time, the JSP file is compiled into servlet source code and the servlet source code is compiled into a servlet class file. This one time compilation step accounts for the delay in displaying the test client page. After the JSP page has been compiled as part of the first access, subsequent requests are processed very quickly and the resulting page appears almost immediately. Force a reload of the page to see how quickly the subsequent next visit is handled. To force a reload, press the shift key and click the reload button on your web browser.

---

2. Click the *Invoke* button next to the create method to create an instance of the GreeterDB bean.

   The create method is under the heading "*Invoke Methods on samples.jdbc.simple.ejb.GreeterDBHome*."

   The web browser displays the new GreeterDB instance as shown below:


Forte For Java Enterprise Java Bean Test Client - Netscape
File Edit View Go Communicator Help
Bookmarks
Location: http://localhost/GreeterDB_TestApp/dispatch.jsp

EJB Navigation

This view allows easy access to instances of the home and remote interfaces of the Enterprise Java Bean undergoing testing. The home interface will be listed at the beginning of the list, instances of the remote interface will follow. Selecting an object here allows methods to be invoked on the object instance. An instance of the tested Bean can be created by selecting the home interface and invoking the appropriate method

- samples.jdbc.simple.ejb.GreeterDBHome
- samples.jdbc.simple.ejb.GreeterDB [7]

Stored Objects

This view provides a list of objects created during your test session which may be used as method parameters. Objects are placed on this list once created from a method invocation or when created directly using the new button provided for complex parameters. Selecting an object from this view allows methods on the object instance to be invoked

samples.jdbc.simple.ejb.GreeterDB [7]
samples.jdbc.simple.ejb.GreeterDBHome

Results of the Last Method Invocation

samples.jdbc.simple.ejb.GreeterDB

Method Invoked: *create ()*
Parameters:
none

Objects may be manipulated using public methods. Method parameters are set directly for primitive types

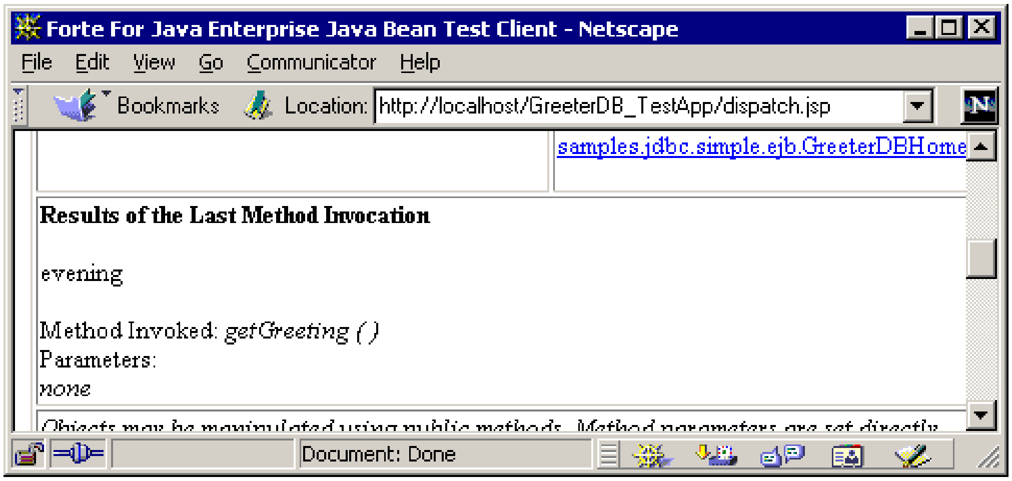
Notice that the `samples.jdbc.simple.ejb.GreeterDB [7]` item is added in the instances list in the upper left hand portion of the page. The item represents the instance of the GreeterDB bean created by invoking the create method on the GreeterDBHome interface.

3. Click the `samples.jdbc.simple.ejb.GreeterDB [7]` instance link.

   This action selects the GreeterDB instance as the EJB module to test. The web browser displays the methods that can be invoked, including the `getGreeting()` business method.

4. Click the *Invoke* button next to the `getGreeting` method.

   The `getGreeting()` method is called, and its return value is listed in the *Results* area:


Forte For Java Enterprise Java Bean Test Client - Netscape
File Edit View Go Communicator Help
Bookmarks
Location: http://localhost/GreeterDB_TestApp/dispatch.jsp
samples.jdbc.simple.ejb.GreeterDBHome

Results of the Last Method Invocation

evening

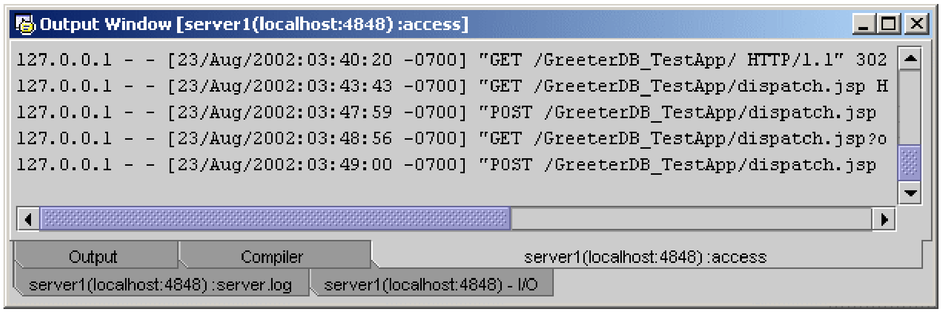
Method Invoked: *getGreeting ()*
Parameters:
none

Objects may be manipulated using public methods. Method parameters are set directly

In the example above, the web browser above shows the word "evening" as the return value. Because the `getGreeting()` method generates a greeting based on the current time, you might see a different greeting value.

5. In the main menu bar of the IDE, select the *Running* tab. In the *Output* window, click the `server1:(localhost:port): access` tab.

A display of the HTTP requests and responses processed by the application server is shown:


Output Window [server1(localhost:4848) :access]

```
127.0.0.1 - - [23/Aug/2002:03:40:20 -0700] "GET /GreeterDB_TestApp/ HTTP/1.1" 302
127.0.0.1 - - [23/Aug/2002:03:43:43 -0700] "GET /GreeterDB_TestApp/dispatch.jsp H
127.0.0.1 - - [23/Aug/2002:03:47:59 -0700] "POST /GreeterDB_TestApp/dispatch.jsp
127.0.0.1 - - [23/Aug/2002:03:48:56 -0700] "GET /GreeterDB_TestApp/dispatch.jsp?o
127.0.0.1 - - [23/Aug/2002:03:49:00 -0700] "POST /GreeterDB_TestApp/dispatch.jsp
```

Output
Compiler
server1(localhost:4848) :access
server1(localhost:4848) :server.log
server1(localhost:4848) - I/O

If you encountered problems while running your EJB, you might normally want to exercise the debugging facilities at this stage. However, for the purposes of this tutorial, we'll defer debugging exercises until we develop and deploy the combined web and EJB application.

Now that you've successfully tested your session bean, proceed to Packaging the Enterprise Bean in an EJB[tm] Module to prepare to include the EJB in an enterprise application.

Packaging the Enterprise Bean in an EJB[tm] Module

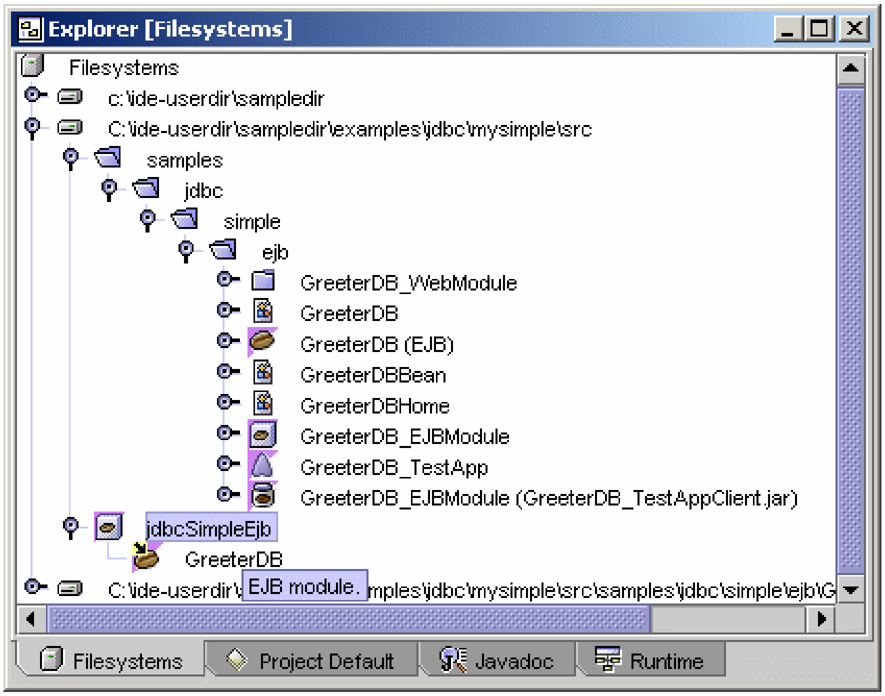
Although the test client automatically packaged your session bean in an EJB module, this module was specifically created for testing purposes. In this section you will package your session bean in a new EJB module that will be used as part of your enterprise application.

To create the EJB module:

1. In the Explorer, select the *Filesystem* tab. (You might need to click the *GUI Editing* tab of the IDE's menu bar to display the Explorer window).
2. Right-click the .../`mysimple/src` filesystem and select *New -> J2EE -> EJB Module*
3. Type `jdbcSimpleEjb` in the *Name* field.
4. Click *Finish*.
5. Right-click the new *jdbcSimpleEjb* EJB module and select *Add EJB*....

   The *Add EJB to EJB Module* browser is displayed.
6. Select the *GreeterDB (EJB)* bean under the `mysimple/src/samples/jdbc/simple/ejb` node
7. Click *OK*.

The GreeterDB session bean is included in the EJB module. Expanding the *jdbcSimpleEjb* node shows the included session bean.



Now that you've prepared the EJB for assembly in an enterprise application, proceed to Creating the Web Application to create the web interface portion of the application.

Creating the Web Application

In this section you will develop the web application by creating the servlet, JavaServer Pages[tm] (JSP[tm]) pages and HTML files and packaging them in a web application module.

- Create the Web Module
- Create Servlets
- Define Servlet Mappings
- Create HTML and JSP Files
- Define EJB Reference

- Define JDBC Resource Reference

Create the Web Module

To create a web module:

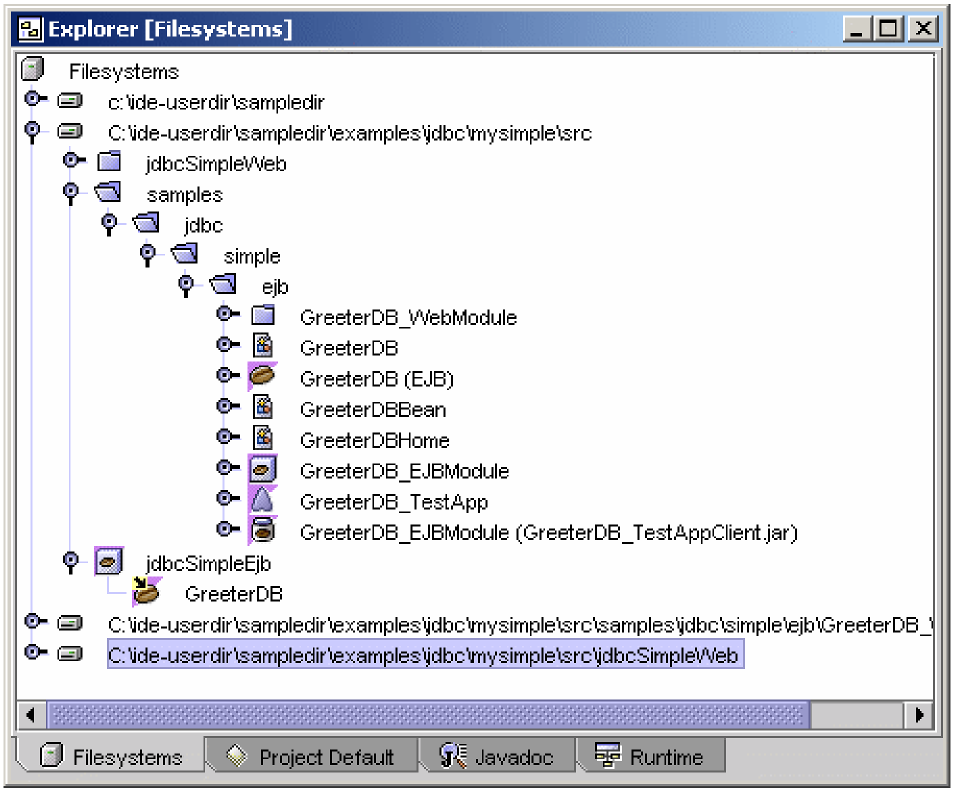
1. In the Explorer window, select the the *Filesystems* node, right click, and select *Mount -> Local Directory*.

   The *New Wizard - Local Directory* dialog window appears.

2. Navigate into the `mysimple/src` directory.
3. Within the `src/` directory, click the *Create New Folder* button.

   A new directory named *New Folder* is created.

4. Rename the new directory to `jdbcSimpleWeb`. To rename the directory, click the *New Folder* directory icon, pause for a few seconds, and click the directory icon again, type `jdbcSimpleWeb`, and press the Enter key.
5. Select the `jdbcSimpleWeb` directory icon and click *Finish*. The new directory is mounted in the Explorer window as a filesystem.



6. Right-click the `mysimple/src/jdbcSimpleWeb` filesystem that you just mounted and select *New -> JSP & Servlet -> Web Module*.

   The *New* wizard appears, showing `mysimple/src/jdbcSimpleWeb` in the *Directory* field.

7. Click *Finish*.

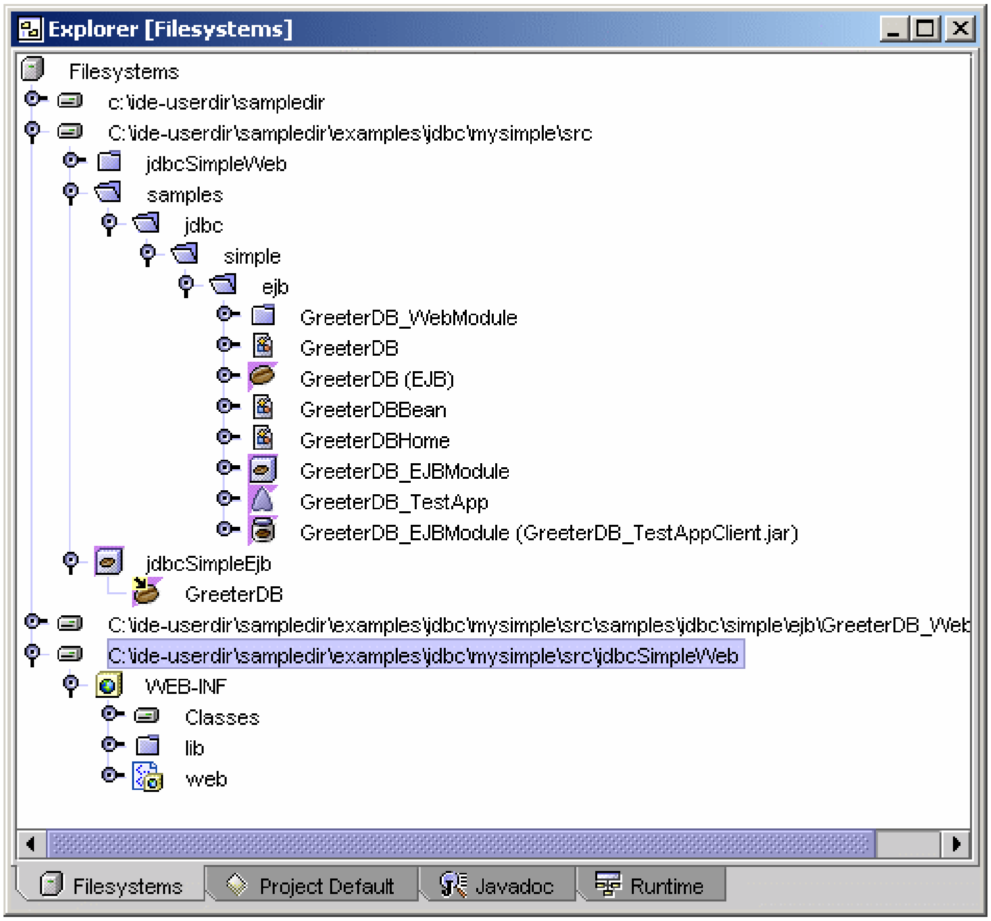
   The *Question* dialog box appears.

8. Click *OK* in the *Question* dialog box. The IDE creates a web module.

   A window might be displayed informing you that the web module is visible in the *Project* tab. If it appears, click *OK* to close the window.

9. Expand the `mysimple/src/jdbcSimpleWeb` filesystem.

   You will see a *WEB-INF* node under the filesystem.



Create Servlets

You will now create two servlets that handle requests. The first servlet, named GreeterDBServlet, requests greetings from the GreeterDB session bean, logs the greeting to the database, and delivers the result via a JSP page. The second servlet, named GreeterDBLogDisplayServlet, retrieves previously generated greetings from the database and displays them via another JSP page.

Create GreeterDBServlet

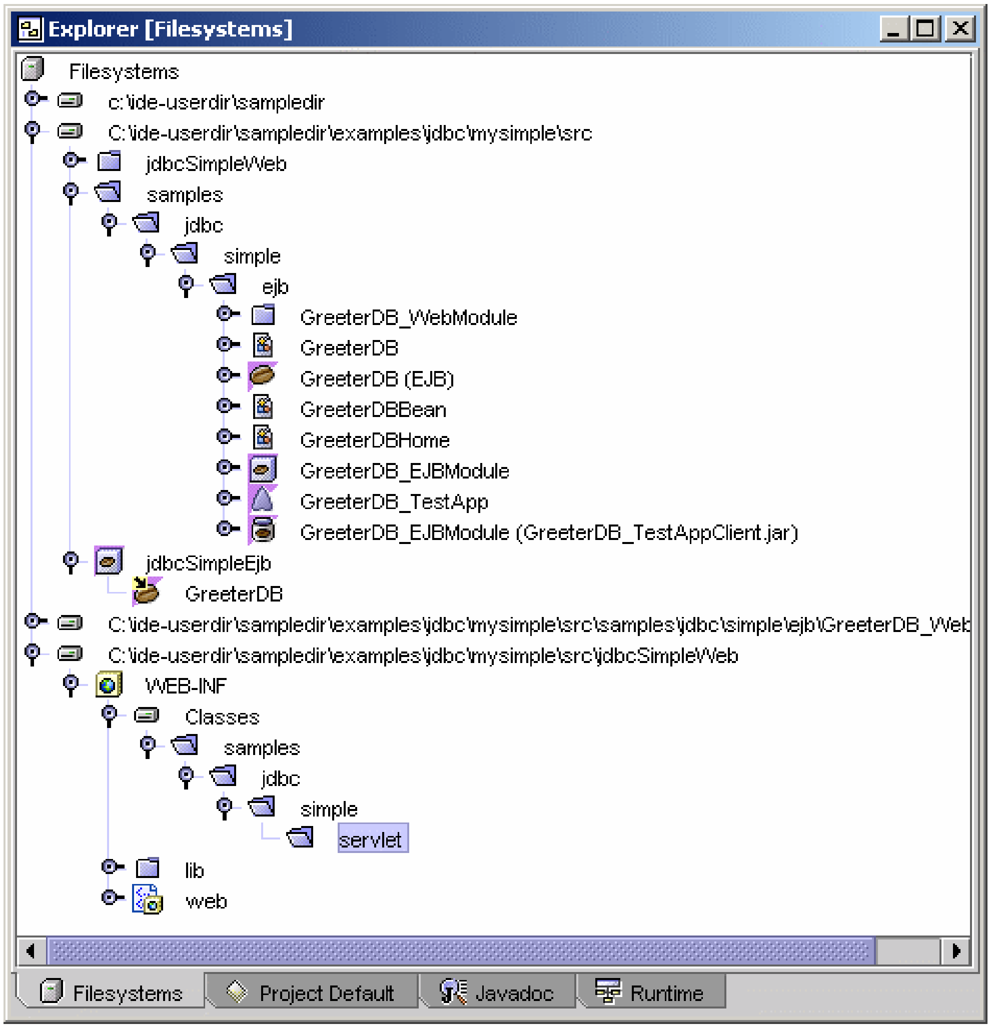
1. Expand the `WEB-INF` folder of the web module filesystem, right-click the `Classes` folder and select *New -> Java Package*.

   The *New Wizard* appears.

2. Type `samples.jdbc.simple.servlet` in the *Name* field and click *Finish*.

   The new `samples.jdbc.simple.servlet` package is created under the `WEB-INF/Classes` folder.

3. Expand the *Classes* folder and its nested folders.



4. Right-click the `servlet` folder and select *New -> JSP Servlet -> Servlet*.

   The *New Wizard* appears.

5. Type `GreeterDBServlet` in the Name field.

6. Click *Finish*.

   The GreeterDBServlet servlet is created in the servlet package and is displayed in the source editor.

7. Add the following `import` statements in bold to the top of the source file below the package statement.

```
package samples.jdbc.simple.servlet;


import javax.servlet.*;
import javax.servlet.http.*;


import java.io.*;
import java.util.*;
import javax.naming.*;
import javax.ejb.*;
import java.sql.*;
import javax.sql.*;
import samples.jdbc.simple.ejb.*;


public class GreeterDBServlet extends HttpServlet {
```

8. Replace the body of the `doGet()` method with the following code in bold:

```
public void doGet (HttpServletRequest
request,HttpServletResponse response)
   throws ServletException, IOException {
   javax.ejb.Handle beanHandle;
   GreeterDBHome myGreeterDBHome;
   GreeterDB myGreeterDBRemote;
   InitialContext initContext = null;
   Hashtable env = new java.util.Hashtable(1);

   System.out.println("\nGreeterDBServlet is executing ...");

   System.out.println("Retrieving JNDI initial context...");
   try {
        initContext = new javax.naming.InitialContext();
        System.out.println("- Retrieved initial context
successfully");
    }
    catch (Exception e) {
        System.out.println("- Exception creating InitialContext: "
+ e.toString());
        return;
    }

    try {
        System.out.println("Looking up dbGreeter bean home
interface...");
        String JNDIName = "java:comp/env/ejb/jdbc-simple";
        System.out.println("- Looking up: " + JNDIName);
        myGreeterDBHome =
(GreeterDBHome)initContext.lookup(JNDIName);
        System.out.println("- Looked up the EJB successfully");
    }
    catch(Exception e) {
        System.out.println("- Greeter bean home not found - " +
        "Is bean registered with JNDI?: " + e.toString());
        return;
    }
    try {
        System.out.println("Creating the dbGreeter bean...");
        myGreeterDBRemote = myGreeterDBHome.create();
        System.out.println("- Created EJB successfully");
    }
    catch(CreateException e) {
        System.out.println("- Could not create the dbGreeter bean:
" + e.toString());
```

```
            return;

        }


        System.out.println("Getting the message from the dbGreeter
bean...");

        String theMessage = myGreeterDBRemote.getGreeting();

        System.out.println("- Got this message from greeter bean: " +
theMessage);


        System.out.println("Getting the name input to this
servlet...");

        String name = request.getParameter("name");

        System.out.println("- Got name: " + name);


        System.out.println("Recording the greeting in the
database...");

        StringBuffer timeStamp = new StringBuffer();

        Calendar rightNow = Calendar.getInstance();

        timeStamp.append(rightNow.get(Calendar.MONTH)+1);
timeStamp.append("/");

        timeStamp.append(rightNow.get(Calendar.DAY_OF_MONTH));
timeStamp.append("/");

        timeStamp.append(rightNow.get(Calendar.YEAR));
timeStamp.append(" ");

        timeStamp.append(rightNow.get(Calendar.HOUR_OF_DAY));
timeStamp.append(":");

        timeStamp.append(rightNow.get(Calendar.MINUTE));
timeStamp.append(":");

        timeStamp.append(rightNow.get(Calendar.SECOND));
timeStamp.append(":");

        timeStamp.append(rightNow.get(Calendar.MILLISECOND));


        StringBuffer query = new StringBuffer("insert into Greeting
(timeStamp,name,message) values ");

        query.append("('");
    query.append(timeStamp.toString()); query.append("','");
    query.append(name); query.append("','");
    query.append("Good " + theMessage); query.append("')");
    try {
        System.out.println("Getting datasource...");
        String dsName = "java:comp/env/jdbc/jdbc-simple";
        DataSource ds =
(javax.sql.DataSource)initContext.lookup(dsName);
        System.out.println("- Got datasource successfully");

        System.out.println("Getting connection...");
        Connection conn = ds.getConnection();
        System.out.println("- Got connection successfully");

        System.out.println("Getting statement...");
        Statement stmt = conn.createStatement();
        System.out.println("- Got statement successfully");

        System.out.println("Executing query...");
        int nRows = stmt.executeUpdate(query.toString());
        System.out.println("- Executed query with result: " +
nRows);

        System.out.println("Closing statement...");
        stmt.close();
        System.out.println("- Closed statement successfully");

        System.out.println("Closing connection...");
        conn.close();
        System.out.println("- Closed connection successfully");
    }
```

```
   catch (Exception ex) {
       System.out.println("- Could not interact with the
database");
       System.out.println("Exception: " + ex.toString());
   }

   System.out.println("Storing the message in request object for
the JSP...");
   request.setAttribute("message", theMessage);
   System.out.println("- Stored message successfully");

   System.out.println("Dispatching JSP for output...");
   response.setContentType("text/html");
   RequestDispatcher dispatcher =
getServletContext().getRequestDispatcher("/GreeterDBView.jsp");
   dispatcher.include(request, response);
   System.out.println("- Dispatched JSP successfully");
   System.out.println("\nGreeterDBServlet is all done\n");
   return;
  }
```

9. Replace the body of the `doPost()` method with the following code in bold:

```
public void doPost (HttpServletRequest
request,HttpServletResponse response)
   throws ServletException, IOException {
   doGet(request,response);
}
```

10. Replace the body of the `getServletInfo()` method with the following code in bold:

```
public String getServletInfo() {
   return "Call a session bean from a servlet" +
          " and deliver result via a JSP." +
          " Log greeting to database.";
}
```

11. Remove the `init()`, `destroy()`, and `processRequest()` methods from the source code.

12. Right-click within the source code window, and select *Compile*.

    When the operation is complete, the word ‘Finished’ appears in the IDE’s Output window.

Now, you will create the second servlet.

Create GreeterDBLogDisplayServlet

1. Using the same procedure as before, create new servlet named `GreeterDBLogDisplayServlet` under the `servlet` folder.

2. Add the following `import` statements in bold to the top of the source file below the package statement.

```
package samples.jdbc.simple.servlet;
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.util.*;
import javax.naming.*;
import javax.ejb.*;
import java.sql.*;
import javax.sql.*;
import samples.jdbc.simple.ejb.*;
public class GreeterDBServlet extends HttpServlet {
```

3. Replace the body of the `doGet()` method with the following code in bold:

```
public void doGet (HttpServletRequest
request,HttpServletResponse response)
   throws ServletException, IOException {
    java.sql.ResultSet rs = null;
    java.sql.Connection conn = null;
    java.sql.Statement stmt = null;
   System.out.println("\nGreeterDBLogDisplayServlet is
executing...");
   try {
      System.out.println("Getting initial context...");
      InitialContext ctx = new InitialContext();
      System.out.println("- Got initial context successfully");

      System.out.println("Getting datasource...");
      String dsName = "java:comp/env/jdbc/jdbc-simple";
      DataSource ds = (javax.sql.DataSource)ctx.lookup(dsName);
      System.out.println("- Got datasource successfully");

      System.out.println("Getting connection...");
      conn = ds.getConnection();
      System.out.println("- Got connection successfully");

      System.out.println("Getting statement...");
      stmt = conn.createStatement();
      System.out.println("- Got statement successfully");

      System.out.println("Executing query...");
      StringBuffer query = new StringBuffer("select * from
Greeting");
      rs = stmt.executeQuery(query.toString());
      System.out.println("- Executed query successfully");
   }
   catch (Exception ex) {
      System.out.println("- Could not interact with the
database");
      System.out.println("- Exception: " + ex.toString());
   }
   System.out.println("Storing the data base resuts in request
object for JSP...");

   request.setAttribute("dbResults", rs);

   System.out.println("Results stored successfully");


   System.out.println("Dispatching JSP for output...");

   response.setContentType("text/html");

   RequestDispatcher dispatcher =
getServletContext().getRequestDispatcher("/GreeterDBLogView.jsp"
);

   dispatcher.include(request, response);

   System.out.println("- Dispatched JSP successfully");

   try {

      System.out.println("Closing statement...");

      stmt.close();

      System.out.println("- Closed statement successfully");


      System.out.println("Closing connection...");

      conn.close();

      System.out.println("- Closed statement successfully");

   }

   catch (Exception ex) {

      System.out.println("- Could not close the statement and
connection");

   }

   System.out.println("\nGreeterDBLogDisplayServlet is all
done\n");

   return;

  }
```

4. Replace the body of the `doPost()` method with the following code in bold:

```
public void doPost (HttpServletRequest
request,HttpServletResponse response)

   throws ServletException, IOException {
```

```
   doGet(request,response);
}
```

5. Replace the body of the `getServletInfo()` method with the following code in bold:

   ```
   public String getServletInfo() {
      return "Retrieve greetings from database and display via a
   JSP.";
   }
   ```

6. Remove the `init()`, `destroy()` and `processRequest()` methods from the source code.
7. Right-click the `GreeterDBLogDisplayServlet` servlet in the Explorer and choose *Compile*.

Define Servlet Mappings

The next step in preparing the servlets for use is to define the names by which the servlets can be referred to in web requests. Servlet mappings are defined at the web module level.

To define the servlet mappings:

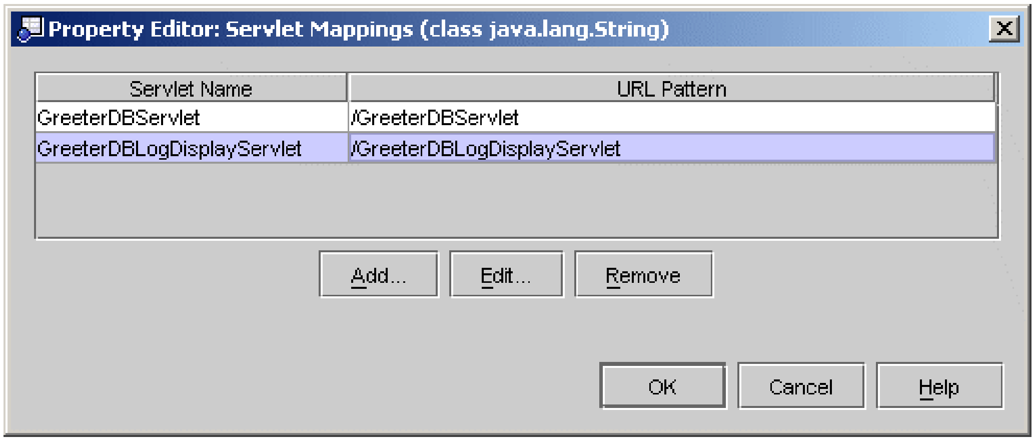
1. Expand the `WEB-INF` folder and select the `web` folder in the Explorer window.

   The `web` folder represents the `web.xml` file, also known as the deployment descriptor, of the web application.
2. Right-click the `web` folder and select *Properties*.

   If the *Properties* window is not already visible, choose *View -> Properties*.
3. Select the *Servlet Mappings* field and then click the ellipsis (...) button.

   The *Servlet Mappings* property editor appears.
4. Select the row with `GreeterDBServlet` in the *Servlet Name* column and click the *Edit...* button.
5. Type `/GreeterDBServlet` in the *URL Pattern* field.
6. Click *OK*.
7. Make the same type of change for the GreeterDBLogDisplayServlet by setting the URL Pattern field to `/GreeterDBLogDisplayServlet`.

The Servlet Mappings property editor should look like this when you are finished modifying the mappings:


Property Editor: Servlet Mappings (class java.lang.String)

Servlet Name	URL Pattern
GreeterDBServlet	/GreeterDBServlet
GreeterDBLogDisplayServlet	/GreeterDBLogDisplayServlet

8. Click *OK* to complete the mappings.

Create HTML and JSP Files

The application consists of a single HTML file and two JSP files. In this step, you will create these files.

To create the HTML page and JSP pages:

1. Right-click the `mysimple/src/jdbcSimpleWeb` filesystem and select *New -> JSP & Servlet -> HTML File*.

   The *New Wizard* appears.

2. Type `index` in the *Name* field

CAUTION **File extensions are added automatically**: Since the IDE automatically adds the appropriate extension to the file name as you create new files such as HTML and JSP files, you should not enter the extension when specifying the name of the page.

If you would like to see the file extensions displayed in the Explorer window, go to the IDE's menu bar, select *Tools -> Options*. Then select *IDE Configuration -> System -> System Settings*. Set *Show File Extensions* to `True`.

---

3. Click *Finish*.

   The `index.html` file is created and is displayed in the source editor.

4. Copy the content of the `index.html` file from the following location and paste it into the source editor, overwriting the existing page.

   ```
   <appserver_install_dir>/samples/jdbc/simple/src/docroot/index.ht
   ml
   ```

5. Right-click the `mysimple/src/jdbcSimpleWeb` filesystem again and select *New -> JSP & Servlet -> JSP*.

   The *New Wizard* appears.

6. Type `GreeterDBView` in the Name field.

7. Click *Finish*.

   The `GreeterDBView.jsp` file is created and is displayed in the source editor.

8. Copy the content of the `GreeterDBView.jsp` file from the following location and paste it into the source editor, overwriting the existing page.

   ```
   <appserver_install_dir>/samples/jdbc/simple/src/docroot/GreeterD
   BView.jsp
   ```

9. Create another JSP file named `GreeterDBLogView` by following the same steps as above, but using the existing `GreeterDBLogView.jsp` file as the source.

Define EJB Reference

Before using JNDI to look up EJB references, you need to set up a reference declaration. The reference declaration maps the reference name (used in the lookup statement) to an actual enterprise bean.

To set up a reference declaration to the referenced session bean:

1. Select the `web` folder in the Explorer, right-click and select *Properties*.

2. Select the *References* tab of the *Properties* window.

3. Select the *EJB References* field and then click the ellipsis (...) button.

   The *EJB References* property editor appears.

4. Click *Add...*.

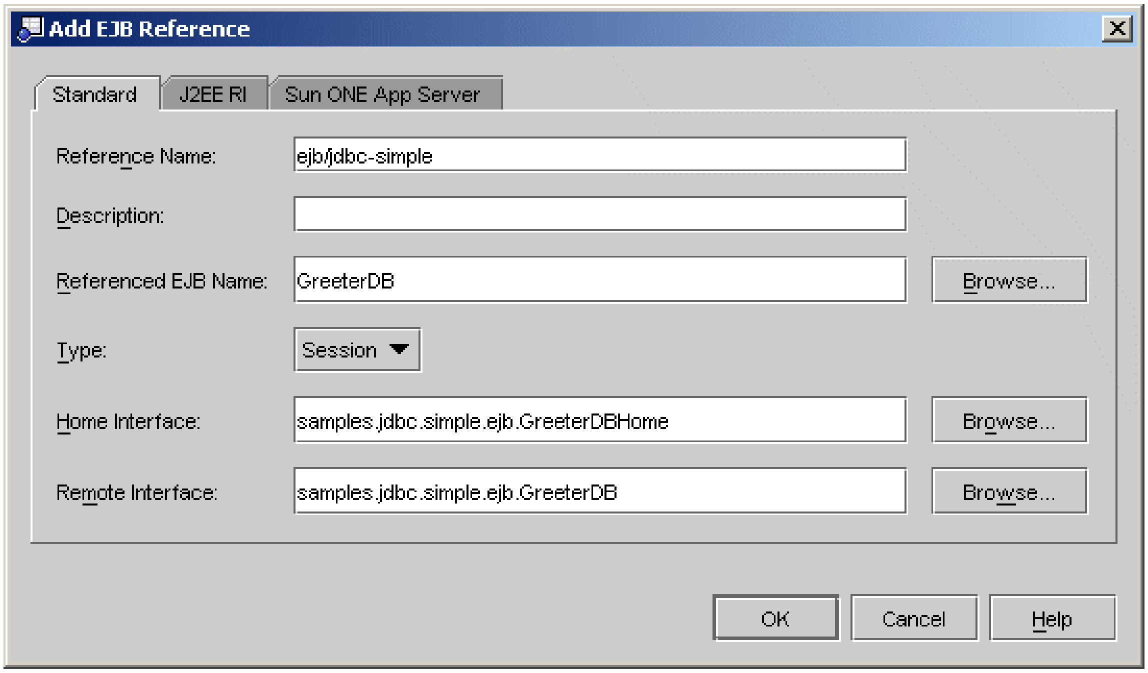
5. Type `ejb/jdbc-simple` in the *Reference Name* field.

6. For the *Referenced EJB Name* field, click the *Browse* button.

   The *Select an EJB browser* is displayed.

7. Select the *GreeterDB (EJB)* bean under the `mysimple/src/samples/jdbc/simple/ejb` folder and click *OK*.

   Notice that the *Home Interface* and *Remote Interface* fields are automatically filled when you select the referenced enterprise bean. The *Add EJB Reference* property editor looks like this:



Reference Name: ejb/jdbc-simple

Description:

Referenced EJB Name: GreeterDB

Type: Session

Home Interface: samples.jdbc.simple.ejb.GreeterDBHome

Remote Interface: samples.jdbc.simple.ejb.GreeterDB

8. Click the *Sun ONE App Server* tab.

9. Enter `ejb/my-jdbc-simple` in the *JNDI Name* field.

CAUTION **Avoiding name conflicts**: If you already deployed the JDBC Simple application while following the application server's Getting Started guide, the JNDI name `ejb/jdbc-simple` will have already been registered in the application server environment. Therefore, it is very important that you specify a slightly modified JNDI name, `ejb/`**`my`**`-jdbc-simple`, for this variant of the JDBC Simple application. Otherwise, JNDI naming conflicts would occur when the application is deployed.

10. Click *OK* to close the *Add EJB References* dialog box.
11. Click *OK* to close the *EJB References* property editor.

Define JDBC Resource Reference

Similar to EJB references, before an application references JDBC resources, these resources must be defined in the module's deployment descriptor.

To add the reference:

1. In the *References* tab of the `web` folder's property window, select the *Resource References* field and then click the ellipsis (...) button.

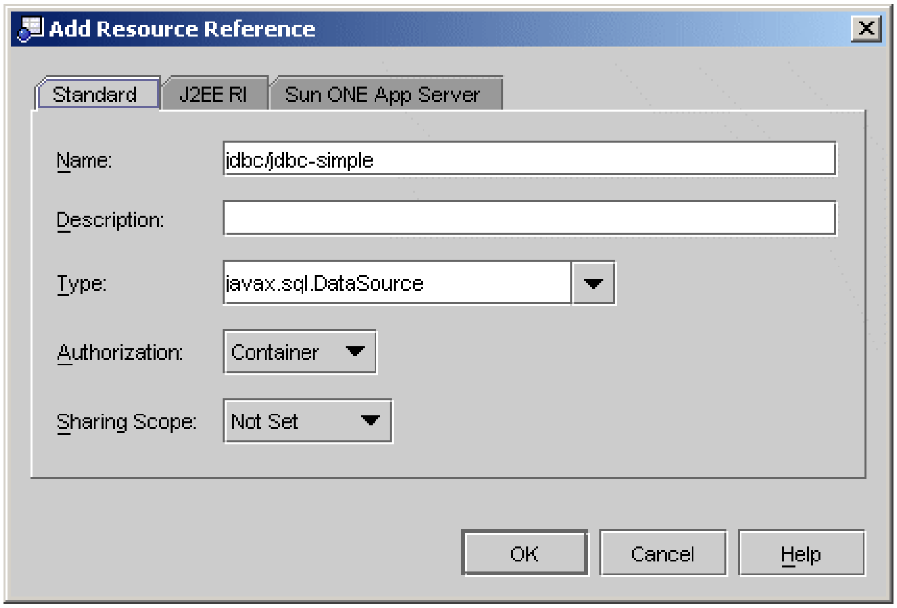
   The *Resource References* property editor appears.

2. Click *Add...*
3. Type `jdbc/jdbc-simple` in the *Name* field.

   This reference name must match the JNDI name used in the servlets.

4. Click the *Sun ONE App Server* tab.
5. Type `jdbc/jdbc-simple` in the *JNDI Name* field.

   This name specifies JDBC resource, defined in the application server's configuration. Later in this tutorial, you will define this resource.


Add Resource Reference

Name: jdbc/jdbc-simple

Description:

Type: javax.sql.DataSource

Authorization: Container

Sharing Scope: Not Set

6. Click *OK* to close the *Add Resource Reference* dialog box.
7. Click *OK* to close the *Resource References* property editor.

Your next step is to combine the EJB and web modules into an enterprise application. Proceed to Assembling the J2EE[tm] Application.

Assembling the J2EE[tm] Application

You have created the EJB[tm] module and web module so far. You will now create a J2EE application that will be deployed to the application server.

To create the J2EE Application:

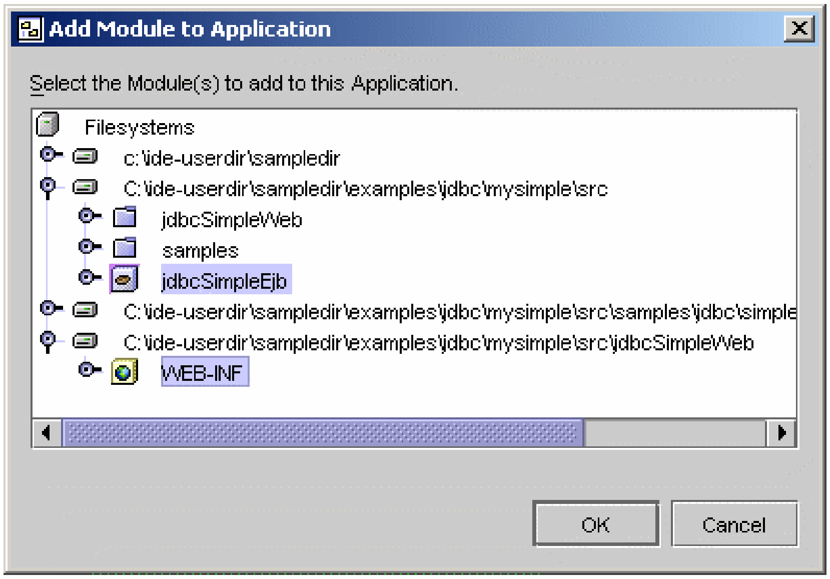
1. Right-click the `mysimple/src filesystem` select *New -> J2EE -> Application*.

   The *New Wizard* is displayed.
2. Type `jdbcSimpleApp` in the *Name* field.
3. Click *Finish*.

4. Right-click the `jdbcSimpleApp` J2EE application node and select *Add Module*....

   The *Add Module* to Application browser is displayed.

5. Using the Control key, select both the *jdbcSimpleEjb* node under the `mysimple/src` filesystem and the `WEB-INF` node under the `mysimple/src/jdbcSimpleWeb` filesystem.

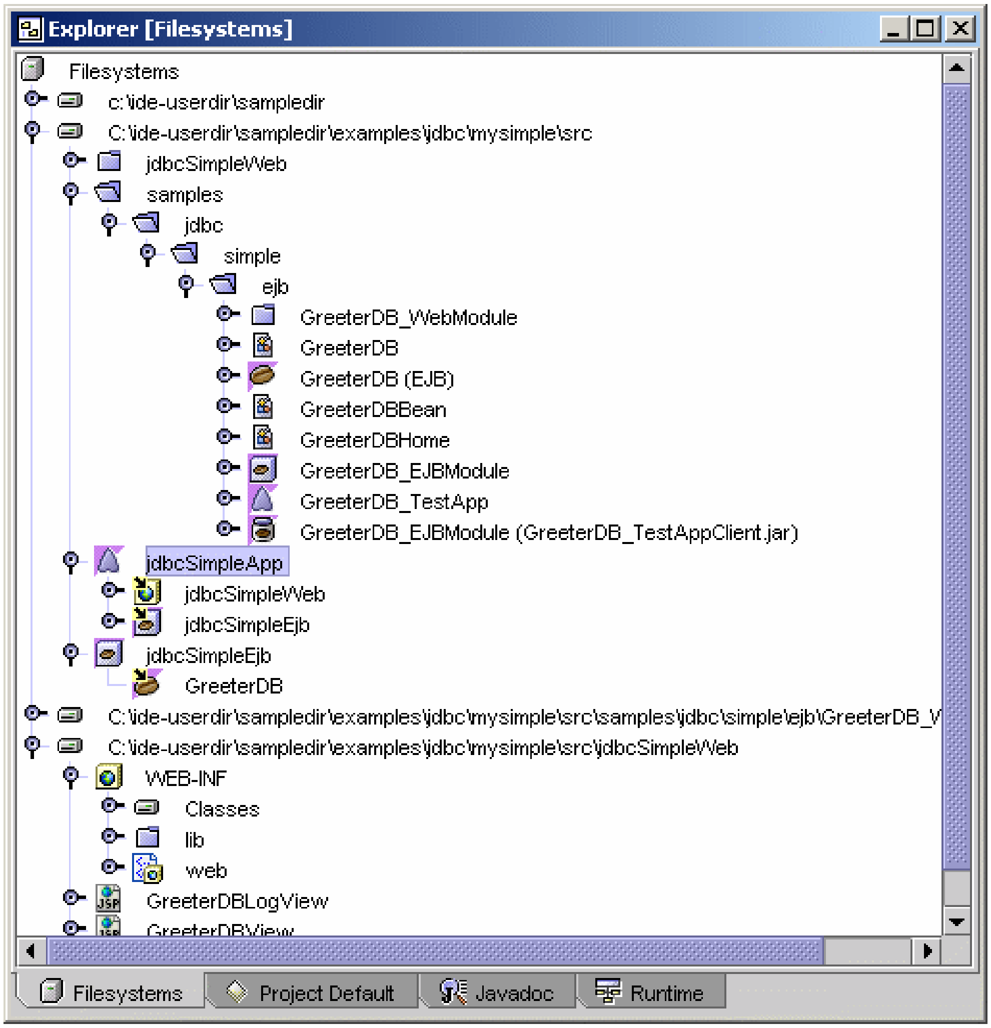

Add Module to Application

Select the Module(s) to add to this Application.

Filesystems
c:\ide-userdir\sampledir
C:\ide-userdir\sampledir\examples\jdbc\mysimple\src
jdbcSimpleWeb
samples
jdbcSimpleEjb
C:\ide-userdir\sampledir\examples\jdbc\mysimple\src\samples\jdbc\simple
C:\ide-userdir\sampledir\examples\jdbc\mysimple\src\jdbcSimpleWeb
WEB-INF

6. Click *OK*.

   If you expand the `jdbcSimpleApp` application node, the two modules are listed:



```
Filesystems
  c:\ide-userdir\sampledir
  C:\ide-userdir\sampledir\examples\jdbc\mysimple\src
    jdbcSimpleWeb
    samples
      jdbc
        simple
          ejb
            GreeterDB_WebModule
            GreeterDB
            GreeterDB (EJB)
            GreeterDBBean
            GreeterDBHome
            GreeterDB_EJBModule
            GreeterDB_TestApp
            GreeterDB_EJBModule (GreeterDB_TestAppClient.jar)
    jdbcSimpleApp
      jdbcSimpleWeb
      jdbcSimpleEjb
    jdbcSimpleEjb
      GreeterDB
  C:\ide-userdir\sampledir\examples\jdbc\mysimple\src\samples\jdbc\simple\ejb\GreeterDB_V
  C:\ide-userdir\sampledir\examples\jdbc\mysimple\src\jdbcSimpleWeb
    WEB-INF
      Classes
      lib
      web
    GreeterDBLogView
    GreeterDBView
```

7. In the IDE's menu bar, select *File -> Save All.*

Before exercising the application, you need to configure the database connectivity for the application. Proceed to Setting Up Database Connectivity.

Setting Up Database Connectivity

Since the enterprise application uses the Java Database Connectivity (JDBC[tm]) API to record greetings to a database, you need to define the necessary JDBC-related settings in the application server environment prior to deploying and exercising the application. JDBC configuration involves setting up a JDBC driver, defining a JDBC connection pool and registering the JDBC resources used by your application.

Once you define the necessary JDBC connection pool and resource entries, you will start the PointBase Server database.

- Configure the JDBC Driver
- Define the JDBC Connection Pool
- Define the JDBC Resource
- Start the PointBase Server Database

Configure the JDBC Driver

A JDBC driver must be configured in the classpath of the application server before you can exercise applications that use JDBC.

Since the PointBase Type 4 JDBC driver should already be configured in your application server instance's environment (based on either the combined installation of the application server with PointBase or through your installation and configuration of PointBase), there is no need for you to perform any additional JDBC driver set up steps.

CAUTION **Solaris 9 Users**: If the application server was installed as part of a Solaris 9 installation, you should have already followed *Getting Started with Sun ONE Application Server 7* on the Sun Microsystems documentation web site. While performing the steps in the Getting Started guide, you should have installed PointBase and configured the application server to use the PointBase JDBC driver. If you have not already done so, make sure that you've completed the Getting Started guide prior to continuing with this tutorial.

Define the JDBC Connection Pool

Before running the sample application, you need to define a suitable JDBC connection pool that maps to the PointBase database server and a JDBC resource that associates the JDBC references made in the sample application to the JDBC connection pool definition.

If you already exercised this sample application as part of the *Getting Started* guide, the JDBC connection pool may already be defined with the application server.

To determine whether or not the connection pool has already been registered:

1. Select the *Runtime* tab of the Explorer window.
2. Expand the *Server Registry*, *Installed Servers*, *Sun ONE Application Server 7* nodes.
3. Expand the `localhost:4848` node followed by the `server1(localhost:`*port*`)` node. Use the appropriate administrative server port number for your environment.
4. Attempt to expand the *Registered JDBC Connection Pool* node to list any registered connection pools.

If you either cannot expand the connection pool node or do not see a connection pool entry named `PointBasePool`, continue with the next section to register the appropriate connection pool.

If you see an entry named `PointBasePool`, scan the following instructions to ensure that the settings of this pool match the requirements of the application.

To register the JDBC connection pool:

1. Right-click the *Unregistered JDBC Connection Pools* node and select *Add New JDBC Connection Pool*.

   The Properties of *JdbcConnectionPool* property sheet is displayed.
2. Type `com.pointbase.jdbc.jdbcDataSource` in the *Datasource Classname*.
3. Type `PointBasePool` in the *Name* field.
4. Select the *Properties* field and then click the ellipsis (...) button.

   A property editor appears without any entries.
5. Add the following sets of values to the property name, value, and description fields, using the *Add* button:

Name	**Value**
DatabaseName	`jdbc:pointbase:server://localhost/sun-appserv-samples`
User	`jdbc`
Password	`jdbc`

CAUTION **Non-default PointBase Server Port**: If you made your own copy of the PointBase Server environment and selected a PointBase server port number other than the default value of 9092, then you need to specify the port number in the *DatabaseName* property.

The *DatabaseName* property specified for the JDBC Connection Pool must be changed from:

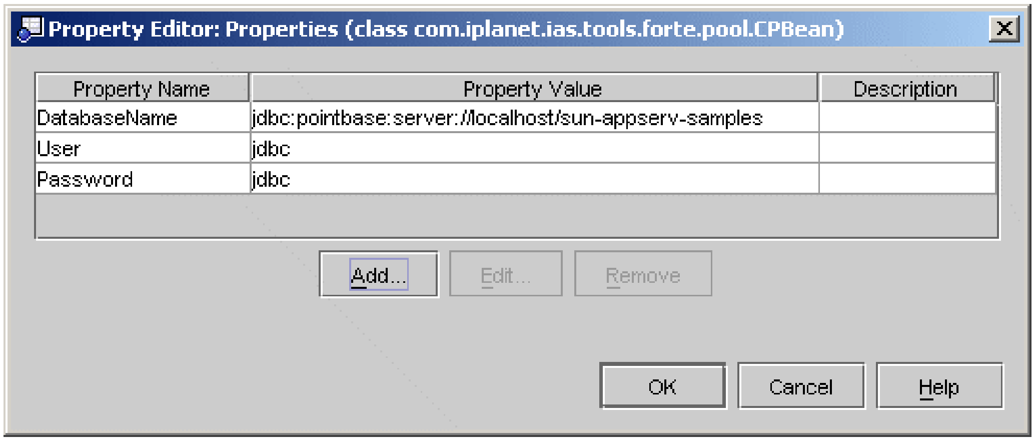
```
jdbc:pointbase:server://localhost/sun-appserv-samples
```

to:

```
jdbc:pointbase:server://localhost:<port>/sun-appserv-samples
```

Where `<port>` is the port number specified in the `StartServer` script for PointBase.

6. Click *OK*.
7. Clear the value of the *Table Name* field.


Property Editor: Properties (class com.iplanet.ias.tools.forte.pool.CPBean)
Add...
Edit...
Remove
OK
Cancel
Help

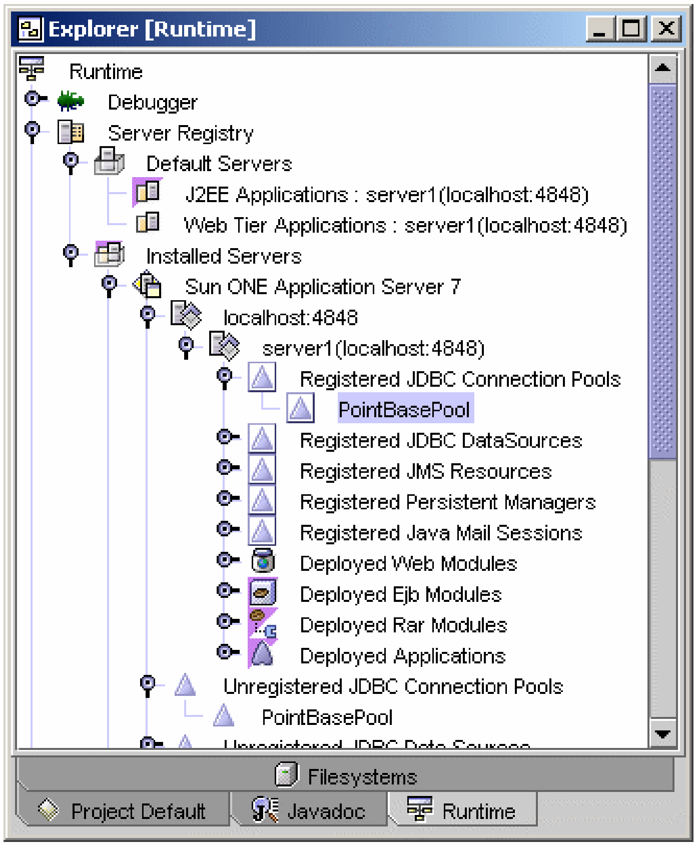
Property Name	Property Value	Description
DatabaseName	jdbc:pointbase:server://localhost/sun-appserv-samples	
User	jdbc	
Password	jdbc	

8. Close the property sheet.
9. Right-click the `PointBasePool` node and choose *Register*.

   A new *Select Server to Register* dialog box appears.
10. Select `server1` and click *Register*.

    An Information dialog box appears indicating a successful registration.
11. Click *OK* to close the *Information* dialog box.
12. Click *OK* to close the *Select Server to Register to* dialog box.
13. Expand the *Registered JDBC Connection Pools* node under the `server1(localhost:4848)` node.

    The registered `PointBasePool` JDBC connection pool is displayed.



If you do not see the `PointBasePool` node under the *Registered JDBC Connection Pools* node, right-click the *Registered JDBC Connection Pools* node and select *Refresh List*.

Define the JDBC Resource

Now that the JDBC connection pool definition has been created, you are ready to define a JDBC resource and associate it with the connection pool entry.

If you already exercised this sample application as part of the *Getting Started with Sun ONE Application Server 7* on the Sun Microsystems documentation web site, a suitable JDBC resource may already be defined with the application server.

To determine whether or not a suitable JDBC resource has already been registered:

1. Expand the `localhost:4848` node followed by the `server1(localhost:`*port*`)` node. Use the appropriate administrative server port number for your environment.

2. Attempt to expand the *Registered JDBC DataSources* node to list the registered JDBC resources.

   If you either cannot expand the JDBC DataSources node or do not see a datasource entry named `jdbc/jdbc-simple`, continue with the next section to register the appropriate resource.

   If you see an entry named `jdbc/jdbc-simple`, scan the following instructions to ensure that the settings of this resource match the requirements of the application.

To register the JDBC resource (referred to as a JDBC "DataSource" in the IDE):

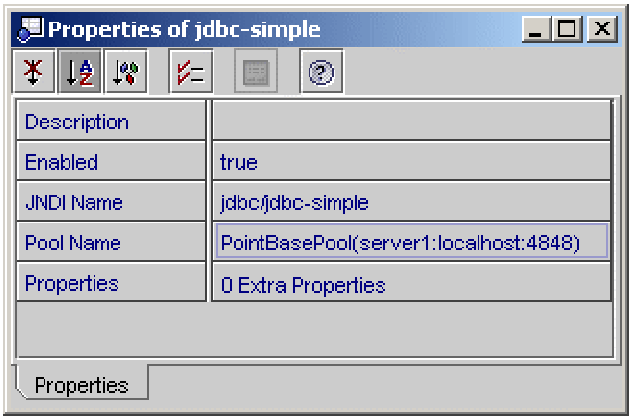
1. Right-click the *Unregistered JDBC DataSources* node and select *Add new Data Source*.

   The Properties of DataSource property sheet is displayed, and a new DataSource node is created under the *Unregistered Data Sources* node

2. Type `jdbc/jdbc-simple` in the *JNDI Name* field.

   The resource name here must match the JNDI name that you have assigned when creating the web application.

3. Select `PointBase(server1:localhost:`*port*`)` in the *Pool Name* field.



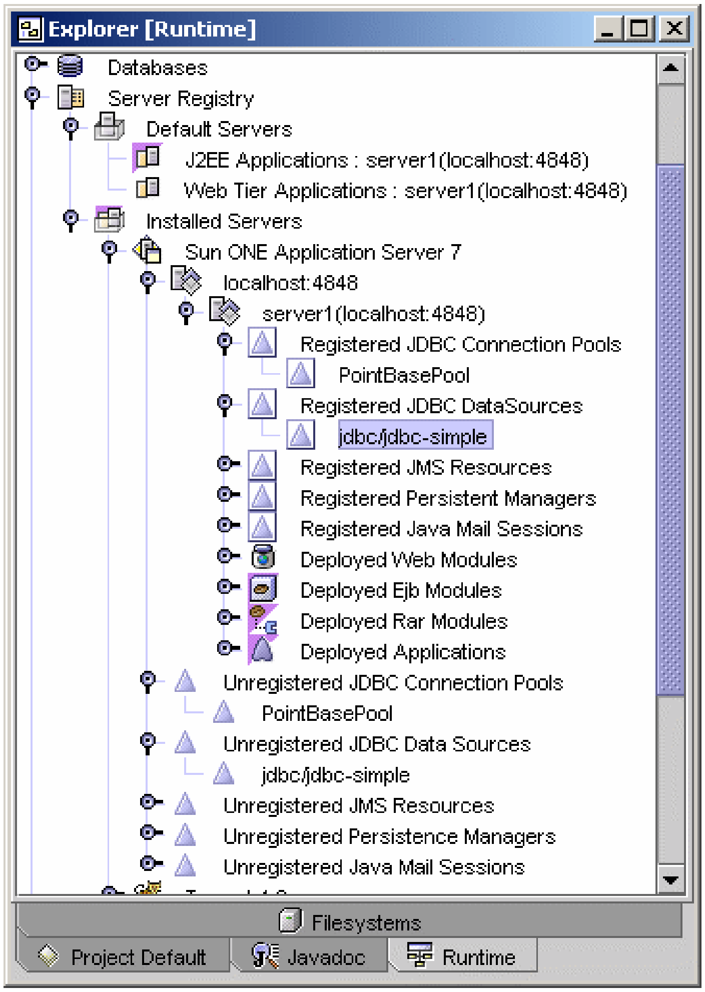
Description	
Enabled	true
JNDI Name	jdbc/jdbc-simple
Pool Name	PointBasePool(server1:localhost:4848)
Properties	0 Extra Properties

4. Close the property sheet.
5. Right-click the new `jdbc-simple` node and choose *Register*...

   A *Select Server to Register to* dialog box is displayed.
6. Select `server1` and click *Register*.

   An Information dialog box appears indicating a success.
7. Click *OK* to close the *Information* dialog box.
8. Click *OK* to close the *Select Server to Register to* dialog box.
9. Expand the Registered *JDBC DataSources* node under the `server1 (localhost:4848)` node.

   The registered `jdbc/jdbc-simple` data source is displayed.


Explorer [Runtime]
Databases
Server Registry
Default Servers
J2EE Applications : server1(localhost:4848)
Web Tier Applications : server1(localhost:4848)
Installed Servers
Sun ONE Application Server 7
localhost:4848
server1(localhost:4848)
Registered JDBC Connection Pools
PointBasePool
Registered JDBC DataSources
jdbc/jdbc-simple
Registered JMS Resources
Registered Persistent Managers
Registered Java Mail Sessions
Deployed Web Modules
Deployed Ejb Modules
Deployed Rar Modules
Deployed Applications
Unregistered JDBC Connection Pools
PointBasePool
Unregistered JDBC Data Sources
jdbc/jdbc-simple
Unregistered JMS Resources
Unregistered Persistence Managers
Unregistered Java Mail Sessions
Filesystems
Project Default
Javadoc
Runtime

Start the PointBase Server Database

The database server can be started by performing one of the following actions:

- Windows Platforms:

  PointBase installed with the application server:

Start -> Programs -> Sun Microsystems -> Sun ONE Application Server 7 -> Start PointBase

PointBase downloaded and installed separately:

Execute: `<pointbase_install_dir>/tools/server/startserver.bat`

- UNIX Platforms:

  PointBase installed with the application server:

  Execute: `<appserver_install_dir>/pointbase/server/StartServer.sh`

  PointBase downloaded and installed separately:

  Execute: `<pointbase_install_dir>/tools/server/startserver.sh`

Once you execute this script, you will see the following text in either a command window or at the UNIX terminal prompt:

```
Server started, listening on port 9092, display level: 0 ...
>
```

You are finally ready to exercise the application. Proceed to Deploying and Running the J2EE[tm] Application.

Deploying and Running the J2EE[tm] Application

In this exercise you will first override the default value assigned to the context root of the web application. Then you will deploy and run the application.

- Disable EJB Test Application
- Define Web Context Root
- Deploy the Application
- Run the Application

Disable EJB Test Application

Since both the EJB Test and the JDBC Simple applications contain the same EJB module, if you attempted to deploy the JDBC Simple application without at least disabling the EJB Test application, a JNDI naming conflict would occur during deployment of the JDBC Simple application. This is because the `<jndi-name>` value of `ejb/my-jdbc-simple` as contained in the EJB module's deployment descriptor is scoped globally throughout the application server instance.

To disable the EJB Test application:

1. In the Explorer, select the *Runtime* tab and drill down to the `server1(localhost:4848)` node.
2. Expand the *Deployed Applications* node, select `GreeterDB_TestApp`, right-click and select *Disable*.

No JNDI naming conflicts will occur with a prior deployment of the JDBC Simple sample application because you changed the JNDI name of the EJB in the second version of this application to a unique value.

Define Web Context Root

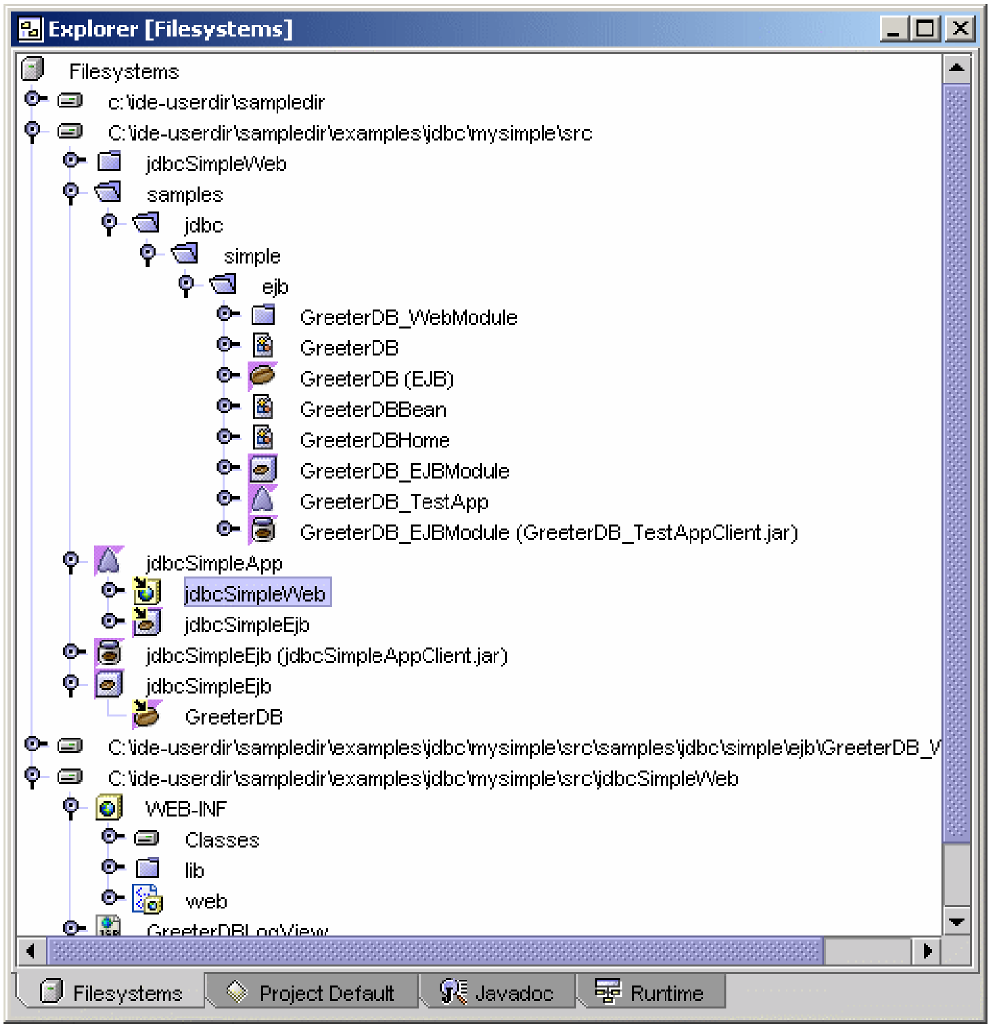
The context root provides a way to distinguish resources of one web application from resources of other web applications deployed to the same server instance. You set the context root on the web module object that is represented in the J2EE application object. In this exercise, you will specify `my-jdbc-simple` as the context root. (If you set the context root to the same value as used by a previously deployed copy of the JDBC Simple sample, an error would occur during deployment because you cannot deploy two web applications with the same context root to the same virtual server).

The context root value determines how end users will access the web application:

`http://`*hostname*`:`*port*`/`**`my-jdbc-simple`**`/`...

To define the context root of the web module:

1. In the Explorer window, under the *Filesystems* tab, expand the `jdbcSimpleApp` node, select the `jdbcSimpleWeb` object, right-click and select *Properties*.



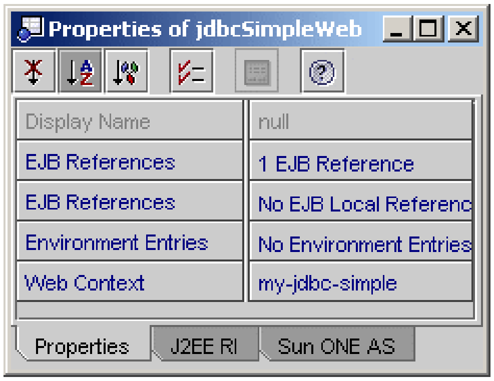
Explorer [Filesystems]

```
Filesystems
  c:\ide-userdir\sampledir
  C:\ide-userdir\sampledir\examples\jdbc\mysimple\src
    jdbcSimpleWeb
    samples
      jdbc
        simple
          ejb
            GreeterDB_WebModule
            GreeterDB
            GreeterDB (EJB)
            GreeterDBBean
            GreeterDBHome
            GreeterDB_EJBModule
            GreeterDB_TestApp
            GreeterDB_EJBModule (GreeterDB_TestAppClient.jar)
    jdbcSimpleApp
      jdbcSimpleWeb
      jdbcSimpleEjb
    jdbcSimpleEjb (jdbcSimpleAppClient.jar)
    jdbcSimpleEjb
      GreeterDB
  C:\ide-userdir\sampledir\examples\jdbc\mysimple\src\samples\jdbc\simple\ejb\GreeterDB_V
  C:\ide-userdir\sampledir\examples\jdbc\mysimple\src\jdbcSimpleWeb
    WEB-INF
      Classes
      lib
      web
    GreeterDBLogView
```

Note that we modify the context setting at the J2EE application level because a single module may be added to different J2EE applications. Each application may require different web context values even though the underlying web application is the same.

2. Type **`my-jdbc-simple`** in the *Web Context* field.

The property sheet looks like this:


Properties of jdbcSimpleWeb

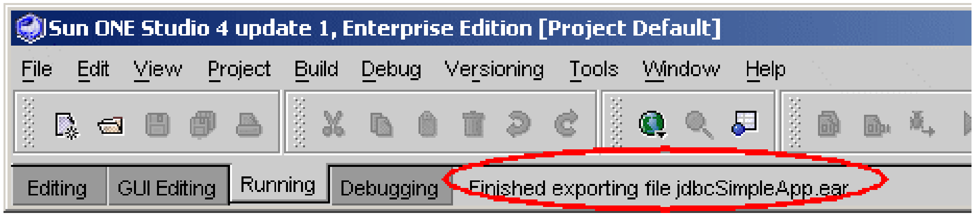
Display Name	null
EJB References	1 EJB Reference
EJB References	No EJB Local Referenc
Environment Entries	No Environment Entries
Web Context	my-jdbc-simple

Deploy the Application

To deploy the application, right-click the `jdbcSimpleApp` node and select *Deploy*.

Several views appear in the *Output* window while a *Progress Monitor* window shows the deployment progress. Wait for the *Progress Monitor* to close automatically.

You can monitor the status of the many operations in the IDE through the status area of the IDE's menu bar. As you deploy the application, you will see status information in this area.



Finished exporting file jdbcSimpleApp.ear

Run the Application

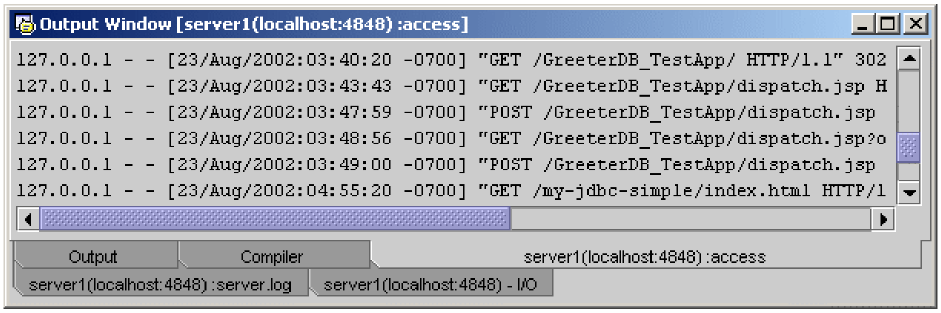
Before running the application, make sure that the PointBase server is running.

To run the application:

1. Open a web browser window and access the following URL:

`http://`*`hostname`*`:`*`port`*`/`**`my`**`-jdbc-simple/index.html`

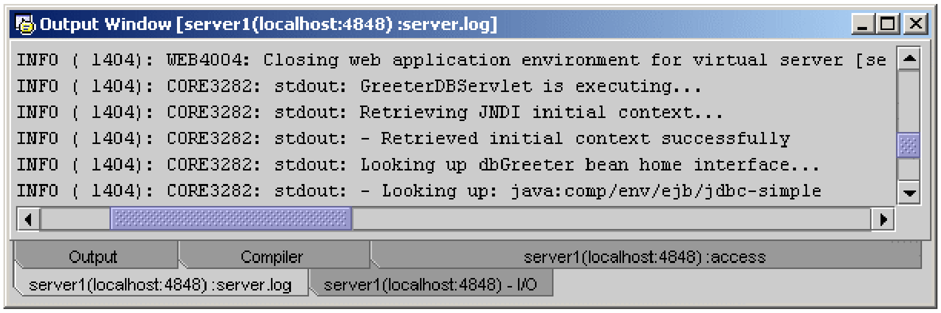
When the web browser connects to the server, the *server1: access view* tab in the output window displays a new line.


Output Window [server1(localhost:4848) :access]
Output
Compiler
server1(localhost:4848) :access
server1(localhost:4848) :server.log
server1(localhost:4848) - I/O

```
127.0.0.1 - - [23/Aug/2002:03:40:20 -0700] "GET /GreeterDB_TestApp/ HTTP/1.1" 302
127.0.0.1 - - [23/Aug/2002:03:43:43 -0700] "GET /GreeterDB_TestApp/dispatch.jsp H
127.0.0.1 - - [23/Aug/2002:03:47:59 -0700] "POST /GreeterDB_TestApp/dispatch.jsp
127.0.0.1 - - [23/Aug/2002:03:48:56 -0700] "GET /GreeterDB_TestApp/dispatch.jsp?o
127.0.0.1 - - [23/Aug/2002:03:49:00 -0700] "POST /GreeterDB_TestApp/dispatch.jsp
127.0.0.1 - - [23/Aug/2002:04:55:20 -0700] "GET /my-jdbc-simple/index.html HTTP/1
```

2. Enter a name and click *Process* in the web browser.

   As the servlet, EJB, and JSP page are executed, the output of `System.out.println` statements are displayed in the *server1: server.log view* tab in the output window.


Output Window [server1(localhost:4848) :server.log]
Output
Compiler
server1(localhost:4848) :access
server1(localhost:4848) :server.log
server1(localhost:4848) - I/O

```
INFO ( 1404): WEB4004: Closing web application environment for virtual server [se
INFO ( 1404): CORE3282: stdout: GreeterDBServlet is executing...
INFO ( 1404): CORE3282: stdout: Retrieving JNDI initial context...
INFO ( 1404): CORE3282: stdout: - Retrieved initial context successfully
INFO ( 1404): CORE3282: stdout: Looking up dbGreeter bean home interface...
INFO ( 1404): CORE3282: stdout: - Looking up: java:comp/env/ejb/jdbc-simple
```

3. Click the *here* link to display a list of the greeting messages generated thus far.

   More log entries appear in the *server.log view* window.

After having run the application successfully, proceed to Modifying the Application to gain a sense of the speed of typical development activities.

Modifying the Application

In this section, you will leverage the dynamic redeployment feature of the application server by modifying several different components in the application, redeploying and retesting the application without restarting the application server. Each exercise is very basic in that you will modify the source code of various types of files, recompile when necessary and simply redeploy the application. The combination of the IDE's automatic reassembly features and the application server's dynamic redeployment capabilities should make your cycle times as short as possible.

- Modify HTML File
- Modify JavaServer Pages[tm] (JSP[tm]) File
- Modify Servlet
- Modify EJB

Modify HTML File

1. In Explorer, double click the `index.html` file located under the web module.
2. Change the following line from:

   ```
   <title>JDBC-SIMPLE Sample Application</title>
   ```

   to:

   ```
   <title>MODIFIED JDBC-SIMPLE Sample Application</title>
   ```

   Also modify the page title that is displayed as part of the HTML content. Change:

   ```
   <b><font face="Arial"><font color="#000000"><font
   size=+2>JDBC-SIMPLE
   Sample Application</font></font></font></b>
   ```

   to:

   ```
   <b><font face="Arial"><font color="#000000"><font
   size=+2>MODIFIED JDBC-SIMPLE
   Sample Application</font></font></font></b>
   ```

3. Save your changes to the file.
4. Select the `jdbcSimpleApp` node in Explorer, right-click and select *Deploy*.
5. After deployment completes, access the application to see the change.

   You should see the modified page title at the top of the browser window.

Modify JavaServer Pages[tm] (JSP[tm]) File

1. In Explorer, double click the `GreeterDBView.jsp` file located under the web module.
2. Change the following line from:

   ```
   Good <%= messageString%>, <%= nameString%>. Enjoy your <%=
   messageString%>.
   ```

   to:

   ```
   Good <%= messageString%>, <%= nameString%>. Have a great <%=
   messageString%>.
   ```

3. Save your changes to the file.
4. Select the `jdbcSimpleApp` node in Explorer, right-click and select *Deploy*.
5. After deployment completes, access the application to see the change.

   Enter your name and click *Process*.

   You should see the modified greeting string.

6. Access the application once more to see how quickly the subsequent requests are processed.

Modify Servlet

1. In Explorer, navigate to the `GreeterDBServlet` file under the *WEB-INF -> Classe*s area of the mounted web module.
2. Double click the `GreeterDBServlet` located under the web module.
3. In the `doGet()` method, change the following line from:

   ```
   System.out.println("\nGreeterDBServlet is executing...");
   ```

   to:

   ```
   System.out.println("\nMODIFIED GreeterDBServlet is
   executing...");
   ```

4. Recompile the servlet by right clicking on the source editor and selecting *Compile*.
5. Select the `jdbcSimpleApp` node in Explorer, right-click and select *Deploy*.

6. After deployment completes, run the application and monitor the server log file. Look for the `MODIFIED` string to appear at the very beginning of the stdout messages generated by the application.

Modify EJB

1. In Explorer, navigate to the `GreeterDBBean` file under the `mysimple/src` filesystem.
2. Double click the `GreeterDBBean` object.
3. In the `getGreeting()` method, change the following line from:

   ```
   System.out.println("GreeterDB EJB is determining message ...");
   ```

   to:

   ```
   System.out.println("MODIFIED GreeterDB EJB is determining message
   ...");
   ```
4. Recompile the implementation class by right clicking on the source editor and selecting *Compile*.
5. Select the `jdbcSimpleApp` node in Explorer, right-click and select *Deploy*.
6. After deployment completes, run the application and monitor the server log file. Look for the `MODIFIED` string to appear in the stdout messages generated by the application.

Now that you've experienced the quick modification cycle, proceed to Debugging the Application to gain a sense of stepping through various components of the application.

Debugging the Application

This section introduces you the ease by which you can debug class files and JavaServer Pages[tm] (JSP[tm]) source files using the IDE and the application server. To start debugging with the application server there is no set up required outside the IDE. All underlying debugging configuration is established automatically by the IDE and the application server.

- Prepare Web Module for Debugging
- Start the Debugger

- Debug the EJB
- Debug a JSP
- Debug a JSP

Prepare Web Module for Debugging

Verify that several web module settings are defined for debugging JSP files. Also set the context root value in the web module.

1. In Explorer, navigate to the `mysimple/src/jdbcSimpleWeb` filesystem.
2. Select the web module's `web.xml` node and display its property sheet.
3. Select *Sun ONE AS* tab.
4. Select the *JSP Param* field and then click the ellipsis (...) button.
5. Verify that *classdebuginfo* and *mappedfile* properties are set to `true`.
6. Click OK to close the property editor.
7. Select the web module's `WEB-INF` node in the `mysimple/src/jdbcSimpleWeb` filesystem and display its property sheet.
8. Enter `/my-jdbc-simple` in the *Context Root* field.

   This value matches the servlet context root value that you assigned while defining the web application.

Start the Debugger

To start the debugger:

1. Choose the `jdbcSimpleApp` J2EE application node.
2. In the IDE's main menu bar, choose *Debug* -> *Start*.

The `server1` instance of the application server restarts, and the IDE switches to the debugging workspace. In addition, the `index.html` page is displayed in the web browser. Note that the application is redpeloyed before the debugging session begins.

CAUTION **Browser does not come up?** Depending on the browser in use, you might have to access the web application through your browser directly.

Debug the EJB

1. Open `GreeterDBBean.java` file in the source editor.
2. Move the cursor to the following line:

   ```
   if(currentHour < 12) {
   ```
3. Right-click and select *Debug -> Toggle Breakpoint*.

   The line in the source editor changes its background color indicating that a breakpoint has been set.
4. In the web browser, type a name, for example `TESTER`, and click *Process*.

   Execution of `GreeterDBBean.java` stops at the breakpoint, and the line is highlighted in another color.
5. In the IDE's main menu bar, select the *Debug* tab.

   In the *Call Stack* view, expand the `samples.jdbc.simple.ejb.GreeterDBBean.getGreeting` node.

   Under the node, you can see the variables declared in the `getGreeting` method. One of the variables is named `currentHour` and it stores the current hour.
6. In the IDE's main menu bar, choose *Debug -> Step Over* a few times until the execution reaches the `System.out.println` statement.

   The value of the message variable changes from null to "morning," "afternoon," or "evening" depending on the value of currentHour variable.
7. Select the `message` variable and display its properties.

   If the properties sheet is not visible, click *Properties ()* button in the *Debug* window.
8. Type `"day"` (including the quotes) in the *Value* field.
9. Select *Debug -> Continue*.

   The debugger continues execution, and the web browser greets `"Good day, TESTER."`

10. Move the cursor to the line with the breakpoint and select `Debug -> Toggle Breakpoint`.

    The breakpoint is removed from the line, and its background color resets to the normal editing color.

Debug a JSP

To debug `GreeterDBView.jsp`:

1. `Open GreeterDBView.jsp` in the source editor.
2. Move the cursor to the following line:

   `Good <%= messageString%>, <%= nameString%>.`
3. Right-click inside the source editor and choose *Compile*.

   You must compile the JSP in the IDE to allow the IDE to correlate between the JSP source file statements and the underlying servlet class file which implements the JSP file.
4. Right-click and select *Debug -> Toggle Breakpoint*.

   The line in the source editor changes its background color indicating that breakpoint has been set.
5. In the web browser, type a name, for example `TESTER`, and click *Process*.

   Execution of the `GreeterDBView.jsp` page stops at the breakpoint, and the line is highlighted indicating that execution stopped at the line.
6. In the *Call Stack* view, expand the `_jasper._GreeterDBViwe_jsp._jspService` node.

   Under the node, you can see the variables for `GreeterDBView.jsp`.
7. Select `nameString` variable.

   Next to the `nameString` variable, the `"TESTER"` string is shown because the name, entered in Step 5, is stored in the `nameString` variable.
8. Type `"HUMAN"` (including the quotes) in the *Value* field.

   The value of `nameString` variable changes to `"HUMAN"`.
9. In the IDE's main menu bar, select *Debug -> Continue*.

The debugger now continues execution of `GreeterDBView.jsp`, and the web browser shows `HUMAN` in the greeting instead of `TESTER`.

10. Move the cursor to the line with the breakpoint and select *Debug -> Toggle Breakpoint*.

    The breakpoint is removed from the line, and its background color resets to the normal editing color.

Stop the Debugger

To stop the debugger:

1. In the IDE's main menu bar, select *Debug -> Finish*.

   The *Finish Debugging Session* dialog box appears.

2. Make sure that all running sessions are checked.
3. Click *OK*.

Once you have completed these exercises, proceed to the final section, Summary.

Chapter 4

Summary

By following this introductory tutorial you explored these key aspects of the Sun ONE Studio 4, Enterprise Edition for Java and Sun ONE Application Server 7:

- Controlling and monitoring the application server through the IDE.
- Defining JDBC connection pools and JDBC resources through the IDE.
- Creation of a session bean using the IDE's EJB Builder.
- Creation and use of an automatically generated EJB test facility.
- Creation of J2EE[tm] Application, EJB and Wed modules.
- Deployment, modification and redeployment of J2EE applications.

Index

SYMBOLS

.nbm 15
_jasper._GreeterDBViwe_jsp._jspService 89

A

About Sun ONE Application Server 7 7
About This Guide 5
  How This Guide is Organized 5
  What you should know 5
Add Admin Server dialog box 21
Add EJB Reference property editor 65
Add New Business Method dialog box 34
Admin Server Host field 21
Admin Server Port field 22
Admin Server Port Number 22
administrative client libraries 12
administrative client support 17
Automatically Generated Methods 33

B

Breakpoint 87, 88
Business Method 34

C

Call Stack view 87, 89
classdebuginfo 86
classpath 70
connection pool, determine if registered 71
context root 79
create a web module 49
Create GreeterDBServlet 52
Create New Folder button 49
Create Servlets 51
create() 42
currentHour 87

D

DatabaseName property 72
Datasource Classname 71
Debug a JSP 88
Debugging 86
Default Servers node 23
delete-domain 22

deploy 81
Deployed Applications node 79
deploying the test application 40
deployment descriptor 62, 66, 79
destroy() 59
Disable EJB Test Application 79
doGet() method 54, 59
domain1 18
doPost() method 58, 61

E

Editing tab 41
EJB 85, 87
EJB Builder facility 30
EJB Reference 64
EJB References field 64
EJB Test Application wizard 38

F

File extensions 63
Filesystem tab 47
Filesystems node 30
Finish Debugging Session dialog box 89
format
  URLs, in manual 6

G

getGreeting() 34, 45
getServletInfo() method 58, 62
Getting Started 5
Getting Started Guide with Sun ONE Application Server 7 5
Getting Started with Sun ONE Application Server 7 5
GreeterDB 29
GreeterDB_EJBModule 38
GreeterDB_TestApp 38
GreeterDB_WebModule 38
GreeterDBBean 85
GreeterDBBean node 33
GreeterDBBean.java 87
GreeterDBLogDisplayServlet 28, 59, 62
GreeterDBServlet 27, 84
GreeterDBView.jsp 64, 84, 88
GUI Editing tab 47

H

handle requests 51
Home Interface 65
HTML 63, 83
HTTP Server's Port Numbe 42

I

IDE 14
import 54
index.html 83
Information dialog box 76
init() 59
Installed Servers 20, 21
Installing the IDE Plugin 11
Invoke button 44
Invoke Methods on
  samples.jdbc.simple.ejb.GreeterDBHome 44

J

J2EE application 67
J2SDK 14
JavaServer Pages 84
JDBC "DataSource" 75

JDBC configuration 70
JDBC Connection Pool 71
JDBC Driver 70
JDBC Resource Reference 66
JDBC resource, determine if registered 75
jdbcSimpleEjb 47
JNDI 32, 64
JNDI Name 75
JNDI Name field 32, 66
JNDI naming conflict 79
JNDI naming conflicts 66
JSP 84
JSP Files 63
JSP Param field 86

L

localhost
    4848 71
    port 21

M

mappedfile 86
message variable 88

N

name conflicts 66
Name field 47
nameString 89
NetBeans 12, 15
New Wizard 63
New Wizard - Local Directory 49
Non-default PointBase Server Port 72

O

Output window 46

P

plugin 11
PointBase Server Database, starting 77
PointBase Server environment 72
PointBase Type 4 JDBC driver 70
PointBasePool 71
port 42
processRequest() method 59
Progress Monitor 81
Properties 20
Properties of DataSource property sheet 75
Properties window 32, 62, 64

Q

Question dialog box 50

R

reference declaration 64
References tab 64
register the JDBC connection pool 71
register the JDBC resource 75
Registered JDBC Connection Pool 71
registered JDBC resources 75
Registration Wizard 19
Remote Development Server 17
Remote Interface field 65
remote server 17
Resource References field 66
Results area 45
Return Type field 34

runide.sh 19
Running tab 46
Runtime 20
Runtime tab 41, 79

S

Save and Save All buttons 34
Select Server to Register dialog box 73
Select Server to Register to dialog box 76
Server Registry 20
Server Status dialog box 24
server.log 40
server1(localhost
    4848) 23, 79
    port) 71
        server.log 40
Servlet 84
Servlet Mappings 62
Servlet Mappings field 62
Servlet Name 62
Session Bean 29
session bean 64
Solaris 21
Solaris 9 13, 70
Start Server Instance button 24
stateless session bean 29
Step 88
Stop the Debugger 89
Sun ONE App Server tab 65
Sun ONE Application Server 7 node 20
Sun ONE Application Server Home property 20
Sun ONE AS tab 32, 86

T

Table Name field 72
Test Client 38, 40

U

Unregistered JDBC Connection Pools 71
Unregistered JDBC DataSources node 75
Update Center Wizard 15
URL Pattern field 62
URLs
    format, in manual 6
user directory 18
User Name field 22
User Password field 22

V

validation 37

W

Web Context field 80
Web Context Root 79
web folder 62
Web Module 49
web module settings 86
web.xml file 62
WEB-INF node 50