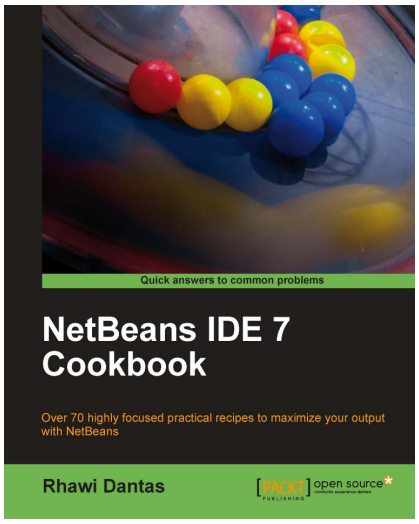
[PACKT] open source
community experience distilled
PUBLISHING

NetBeans IDE 7 Cookbook

Rhawi Dantas



Chapter No. 7
"EJB Application"

In this package, you will find:

A Biography of the author of the book

A preview chapter from the book, Chapter NO.7 "EJB Application"

A synopsis of the book's content

Information on where to buy this book

About the Author

Rhawi Dantas is a Brazilian Software Engineer, more specifically from Recife, with severeal years of experience in the Java platform. He has mainly focused on Web/Server developmen and hsa contributed to projects ranging from mobile/server integration, different customization of IDEs and development of CRMs. He currently works at Paf.com with Java Web development.

He graduated as Bachelor in Information Systems and at that time he had the opportunity to work as a tutor for the University with Object Oriented Programming subject. Besides full-time work he is on his way with his Masters in Software Systems at the Tampere University of Technology.

He is also certified as SCJP, SCWCD, and SCSNI.

This is a small thank you to the three most important women in my life: Sônia Dantas, Paula Mäkinen-Dantas, and Maria Dantas. I would also like to thank the work of my editors, especially Jovita Pinto and Roger D'Souza, and all of the reviewers for their valuable contribution.

NetBeans IDE 7 Cookbook

Welcome to the NetBeans Cookbook.

NetBeans is a Java Integrated Development Environment, IDE, which enables fast application development with the most adopted frameworks, technologies, and servers.

Different than other IDEs, NetBeans comes already pre-packaged with a wide range of functionality out of the box, such as support for different frameworks, servers, databases, and mobile development.

This book does require a minimal knowledge of Java platform, more specifically the language ifself. But the book might as well be used by either beginners, who are trying to dip their toes in new technology, or more experienced developers, who are trying to switch from other IDEs but want to decrease their learning curve of a new environment. NetBeans integrates so many different technologies, many of which are present in this book, that it is beyond the scope of this book to cover all of them in depth. We provide the reader with links and information where to go when further knowledge is required.

What This Book Covers

Chapter 1, NetBeans Head First introduces the developer to the basics of NetBeans by creating basic Java projects and importing Eclipse or Maven projects.

Chapter 2, Basic IDE Usage covers the creation of packages, classes, and constructors, as well as some usability feature.

Chapter 3, Designing Desktop GUI Applications goes through the process of creating a desktop application, then connecting it to a database and even modifying it to look more professional.

Chapter 4, JDBC and NetBeans helps the developer to setup NetBeans with the most common database systems on the market and shows some of the functionality built-in to NetBeans for handling SQL.

Chapter 5, Building Web Applications introduces the usage of web frameworks such as JSF, Struts, and GWT.3

Chapter 6, Using JavaFX explains the basic of JavaFX application states and connecting our JavaFX app to a web service interface.

Chapter 7, EJB Application goes through the process of building an EJB application which supports JPA, stateless, and stateful beans and sharing a service through a web service interface.

Chapter 8, Mobile Development teaches how to create your own CLDC or CDC applications with the help of NetBeans Visual Mobile Designer.

Chapter 9, Java Refactoring lets NetBeans refactor your code to extract classes, interfaces, encapsulate fields, and other options.

Chapter 10, Extending the IDE includes handy examples on how to create your own panels and wizards so you can extend the functionality of the IDE.

Chapter 11, Profiling and Testing covers NetBeans Profiler, HTTP Monitor, and integration with tools that analyze code quality and load generator.

Chapter 12, Version Control shows how to configure NetBeans to be used with the most common version control systems on the market.

7
EJB Application

In this chapter, we will cover:

- Creating an EJB project
- Adding JPA support
- Creating Stateless Session Bean
- Creating Stateful Session Bean
- Sharing a service through Web Service
- Creating a Web Service client

Introduction

Enterprise Java Beans (EJB) is a framework of server-side components that encapsulates business logic.

These components adhere to strict specifications on how they should behave. This ensures that vendors who wish to implement EJB-compliant code must follow conventions, protocols, and classes ensuring portability.

The EJB components are then deployed in EJB containers, also called **application servers**, which manage persistence, transactions, and security on behalf of the developer.

If you wish to learn more about EJBs, visit `http://jcp.org/en/jsr/detail?id=318` or `https://www.packtpub.com/developer-guide-for-ejb3/book`.

For our EJB application to run, we will need the application servers.

Application servers are responsible for implementing the EJB specifications and creating the perfect environment for our EJBs to run in.

Some of the capabilities supported by EJB and enforced by Application Servers are:

- Remote access
- Transactions
- Security Scalability

NetBeans 6.9, or higher, supports the new Java EE 6 platform, making it the only IDE so far to bring the full power of EJB 3.1 to a simple IDE interface for easy development.

NetBeans makes it easy to develop an EJB application and deploy on different Application Servers without the need to over-configure and mess with different configuration files. It's as easy as a project node right-click.

Creating EJB project

In this recipe, we will see how to create an EJB project using the wizards provided by NetBeans.

Getting ready

It is required to have NetBeans with Java EE support installed to continue with this recipe.

If this particular NetBeans version is not available in your machine, then you can download it from `http://download.netbeans.org`.

There are two application servers in this installation package, Apache Tomcat or GlassFish, and either one can be chosen, but at least one is necessary.

In this recipe, we will use the GlassFish version that comes together with NetBeans 7.0 installation package.

How to do it...

1. Lets create a new project by either clicking **File** and then **New Project**, or by pressing *Ctrl+Shift+N*.
2. In the **New Project** window, in the categories side, choose **Java Web** and in **Projects side**, select **WebApplication**, then click **Next**.
3. In **Name and Location**, under Project Name, enter **EJBApplication**.
4. Tick the **Use Dedicated Folder for Storing Libraries** option box.
5. Now either type the folder path or select one by clicking on **browse**.
6. After choosing the folder, we can proceed by clicking **Next**.

7. In **Server and Settings**, under **Server**, choose **GlassFish Server 3.1**.
8. Tick **Enable Contexts and Dependency Injection**.
9. Leave the other values with their default values and click **Finish**.

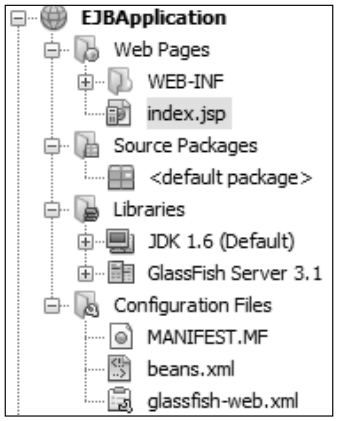
The new project structure is created.

How it works...

NetBeans creates a complete file structure for our project.

It automatically configures the compiler and test libraries and creates the GlassFish deployment descriptor.

The deployment descriptor filename specific for the GlassFish web server is `glassfish-web.xml`.



Adding JPA support

The **Java Persistence API** (**JPA**) is one of the frameworks that equips Java with object/relational mapping. Within JPA, a query language is provided that supports the developers abstracting the underlying database.

With the release of JPA 2.0, there are many areas that were improved, such as:

- Domain Modeling
- EntityManager
- Query interfaces
- JPA query language and others

We are not going to study the inner workings of JPA in this recipe. If you wish to know more about JPA, visit `http://jcp.org/en/jsr/detail?id=317` or

`http://download.oracle.com/javaee/5/tutorial/doc/bnbqa.html`.

NetBeans provides very good support for enabling your application to quickly create entities annotated with JPA.

In this recipe, we will see how to configure your application to use JPA. We will continue to expand the previously-created project.

Getting ready

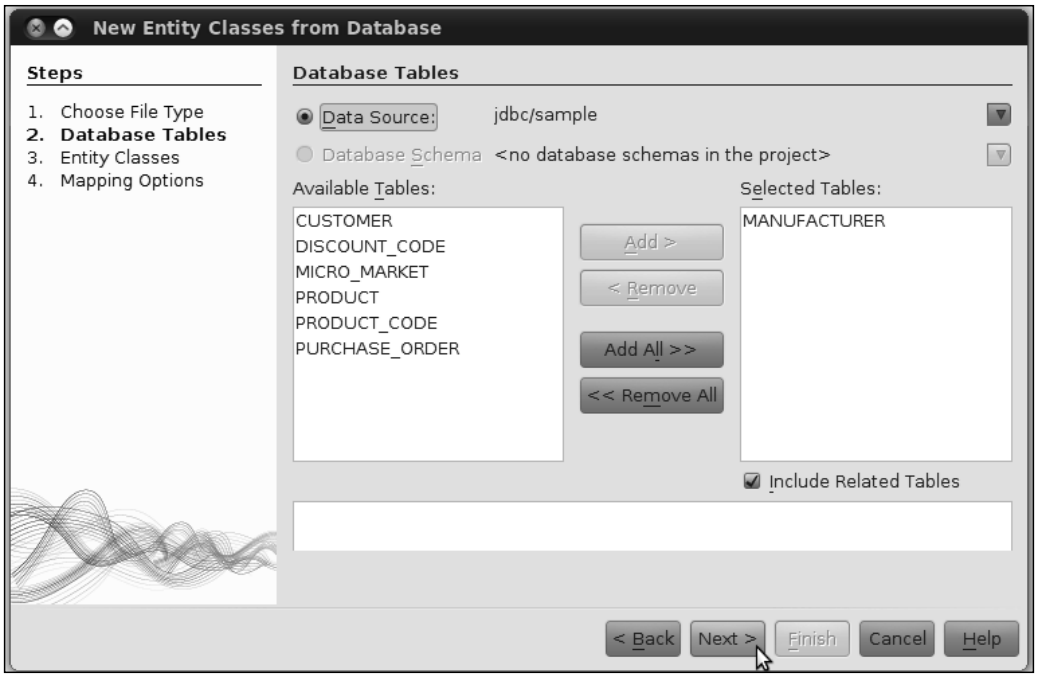
We will use GlassFish Server in this recipe since it is the only server that supports Java EE 6 at the moment.

We also need to have Java DB configured. GlassFish already includes a copy of Java DB in its installation folder. Another source of installed Java DB is the JDK installation directory. If you wish to learn how to configure Java DB, please refer to *Chapter 4*, *JDBC and NetBeans*.

It is not necessary to build on top of the previous recipe, but it is imperative to have a database schema. Feel free to create your own entities by following the steps presented in this recipe.

How to do it...

1. Right-click on **EJBApplication** node and select **New Entity Classes from Database...**.
2. In **Database Tables**: Under **Data Source**, select **jdbc/sample** and let the IDE initialize Java DB.
3. When **Available Tables** is populated, select **MANUFACTURER**, click **Add**, and then click **Next**.



4. In **Entity Classes**: leave all the fields with their default values and only in **Package**, enter entities and click **Finish**.

How it works...

NetBeans then imports and creates our Java class from the database schema, in our case the `Manufacturer.java` file placed under the entities package.

Besides that, NetBeans makes it easy to import and start using the entity straightaway. Many of the most common queries, for example find by name, find by zip, and find all, are already built into the class itself.

The JPA queries, which are akin to normal SQL queries, are defined in the entity class itself. Listed below are some of the queries defined in the entity class `Manufacturer.java`:

```
@Entity
@Table(name = "MANUFACTURER")
@NamedQueries({
    @NamedQuery(name = "Manufacturer.findAll", query = "SELECT m FROM
Manufacturer m"),
@NamedQuery(name = "Manufacturer.findByManufacturerId", query
= "SELECT m FROM Manufacturer m WHERE m.manufacturerId =
:manufacturerId"),
```

The `@Entity` annotation defines that this class, `Manufacturer.java`, is an entity and when followed by the `@Table` annotation, which has a name parameter, points out the table in the Database where the information is stored.

The `@NamedQueries` annotation is the place where all the NetBeans-generated JPA queries are stored. There can be as many `@NamedQueries` as the developer feels necessary. One of the NamedQueries we are using in our example is named `Manufacturer.findAll`, which is a simple select query. When invoked, the query is translated to:

```
SELECT m FROM Manufacturer m
```

On top of that, NetBeans implements the equals, hashCode, and toString methods. Very useful if the entities need to be used straight away with some collections, such as HashMap.

Below is the NetBeans-generated code for both hashCode and the toString methods:

```
    @Override
    public int hashCode() {
        int hash = 0;
        hash += (manufacturerId != null ? manufacturerId.hashCode() :
0);
        return hash;
    }

    @Override
    public boolean equals(Object object) {
        // TODO: Warning - this method won't work in the case the id
fields are not set
        if (!(object instanceof Manufacturer)) {
            return false;
        }
        Manufacturer other = (Manufacturer) object;
        if ((this.manufacturerId == null && other.manufacturerId
!= null) || (this.manufacturerId != null && !this.manufacturerId.
equals(other.manufacturerId))) {
            return false;
        }
        return true;
    }
```

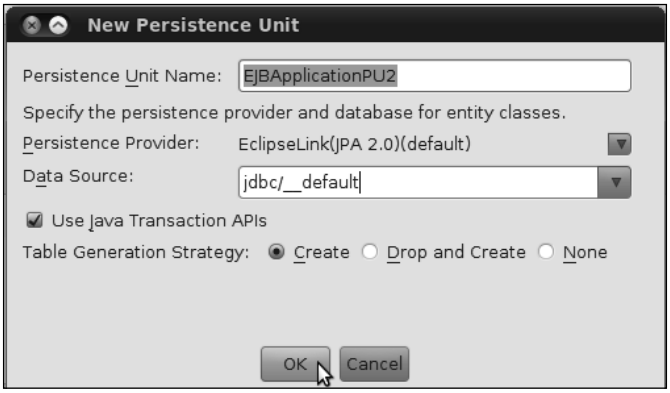
NetBeans also creates a `persistence.xml` and provides a Visual Editor, simplifying the management of different Persistence Units (in case our project needs to use more than one); thereby making it possible to manage the `persistence.xml` without even touching the XML code. A persistence unit, or `persistence.xml`, is the configuration file in JPA which is placed under the configuration files, when the NetBeans view is in Projects mode. This file defines the data source and what name the persistence unit has in our example:

```
<persistence-unit name="EJBApplicationPU" transaction-type="JTA">
  <jta-data-source>jdbc/sample</jta-data-source>
  <properties/>
</persistence-unit>
```

The `persistence.xml` is placed in the configuration folder, when using the Projects view. In our example, our persistence unit name is `EJBApplicationPU`, using the `jdbc/sample` as the data source.

To add more PUs, click on the **Add** button that is placed on the uppermost right corner of the Persistence Visual Editor.

This is an example of adding another PU to our project:



Creating Stateless Session Bean

A Session Bean encapsulates business logic in methods, which in turn are executed by a client. This way, the business logic is separated from the client.

Stateless Session Beans do not maintain state. This means that when a client invokes a method in a Stateless bean, the bean is ready to be reused by another client. The information stored in the bean is generally discarded when the client stops accessing the bean.

This type of bean is mainly used for persistence purposes, since persistence does not require a conversation with the client.

It is not in the scope of this recipe to learn how Stateless Beans work in detail. If you wish to learn more, please visit:

`http://jcp.org/en/jsr/detail?id=318`

or

`https://www.packtpub.com/developer-guide-for-ejb3/book`

In this recipe, we will see how to use NetBeans to create a Stateless Session Bean that retrieves information from the database, passes through a servlet and prints this information on a page that is created on-the-fly by our servlet.

Getting ready

It is required to have NetBeans with Java EE support installed to continue with this recipe.

If this particular NetBeans version is not available in your machine, please visit `http://download.netbeans.org`.

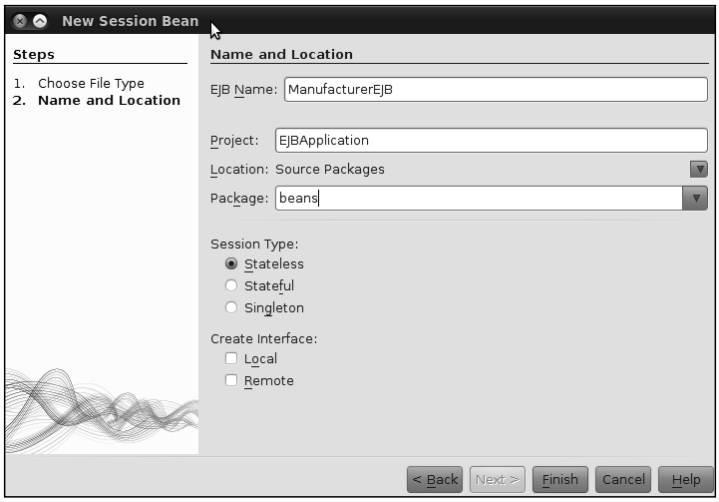
We will use the GlassFish Server in this recipe since it is the only Server that supports Java EE 6 at the moment.

We also need to have Java DB configured. GlassFish already includes a copy of Java DB in its installation folder. If you wish to learn how to configure Java DB refer to the *Chapter 4*, *JDBC and NetBeans*.

It is possible to follow the steps on this recipe without the previous code, but for better understanding we will continue to build on the top of the previous recipes source code.

How to do it...

1. Right-click on **EJBApplication** node and select **New and Session Bean...**.
2. For **Name and Location**: Name the EJB as **ManufacturerEJB**.
3. Under **Package**, enter **beans**.
4. Leave **Session Type** as **Stateless**.
5. Leave **Create Interface** with nothing marked and click **Finish**.



Here are the steps for us to create business methods:

1. Open ManufacturerEJB and inside the class body, enter:

```
@PersistenceUnit
EntityManagerFactory emf;

public List findAll(){
 return emf.createEntityManager().createNamedQuery("Manufacturer.
findAll").getResultList();
}
```

2. Press *Ctrl+Shift+I* to resolve the following imports:

```
java.util.List;
javax.persistence.EntityManagerFactory;
javax.persistence.PersistenceUnit;
```

Creating the Servlet:

1. Right-click on the **EJBApplication** node and select **New and Servlet...**.
2. For **Name and Location**: Name the servlet as **ManufacturerServlet**.
3. Under **package**, enter **servlets**.
4. Leave all the other fields with their default values and click **Next**.
5. For **Configure Servlet Deployment**: Leave all the default values and click **Finish**.

With the **ManufacturerServlet** open:

After the class declaration and before the `processRequest` method, add:

```
@EJB
ManufacturerEJB manufacturerEJB;
```

Then inside the **processRequest** method, first line after the `try` statement, add:

```
List<Manufacturer> l = manufacturerEJB.findAll();
```

Remove the /* TODO output your page here and also */.

And finally replace:

```
out.println("<h1>Servlet ManufacturerServlet at " + request.
getContextPath () + "</h1>");
```

With:

```
for(int i = 0; i < 10; i++ )
 out.println("<b>City</b> " + l.get(i).getCity() + ", <b>State</b> " +
l.get(i).getState() +"<br>" );
```

Resolve all the import errors and save the file.

How it works...

To execute the code produced in this recipe, right-click on the **EJBApplication** node and select **Run**.

When the browser launches append to the end of the URL/ManufacturerServlet, hit *Enter*.

Our application will return City and State names.

One of the coolest features in Java EE 6 is that usage of `web.xml` can be avoided if annotating the servlet. The following code does exactly that:

```
@WebServlet(name="ManufacturerServlet", urlPatterns={"/
ManufacturerServlet"})
```

Since we are working on Java EE 6, our Stateless bean does not need the daunting work of creating interfaces, the `@Stateless` annotation takes care of that, making it easier to develop EJBs.

We then add the persistence unit, represented by the EntityManagerFactory and inserted by the `@PersistenceUnit` annotation.

Finally we have our business method that is used from the servlet. The `findAll` method uses one of the named queries from our entity to fetch information from the database.

Creating Stateful Session Beans

If Stateless Session Beans do not maintain state, it is easy to guess what Stateful Session Beans do. Yes, they maintain the state.

When a client invokes a method in a stateful bean, the variables (state) of that request are kept in the memory by the bean. When more requests come in, the container makes sure that the same bean is used for the same client. This type of bean is useful when multiple requests are required and several steps are necessary for completing a task.

Stateful Beans also enjoy the ease of development introduced by Java EE 6, meaning that they can be created by annotating a POJO with `@Stateful`.

It is not in the scope of this recipe to learn how Stateful Beans work in detail. If you wish to learn more, please visit:

`http://jcp.org/en/jsr/detail?id=318`

Or

`https://www.packtpub.com/developer-guide-for-ejb3/book`

In this recipe, we will see how to use NetBeans to create a stateful session bean that holds a counter of how many times a request for a method was executed.

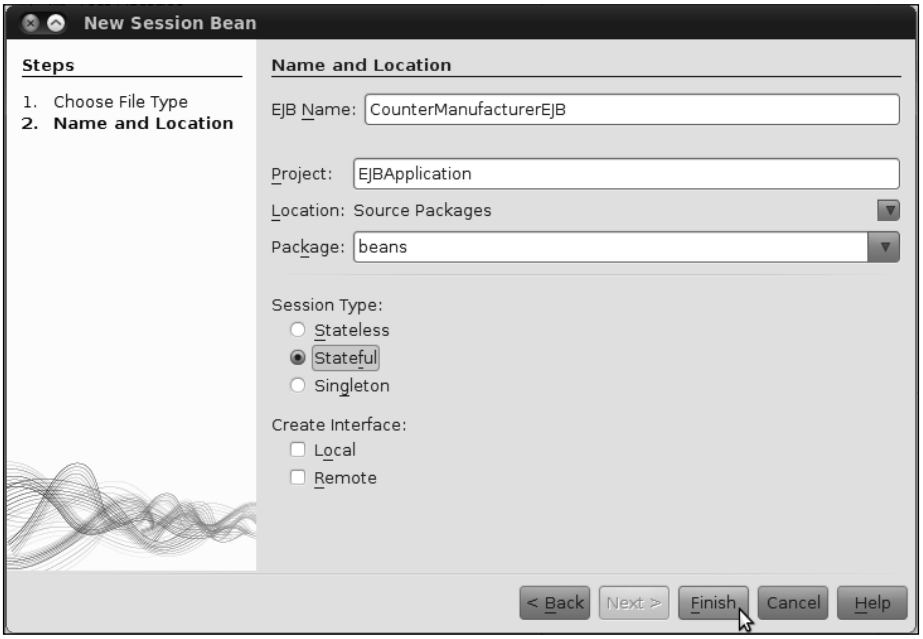
Getting ready

Please find the software requirements and configuration instructions for this recipe in the first *Getting ready* section of this chapter.

This recipe builds on the sources of the previous recipes.

How to do it...

1. Right-click on the EJBApplication node and select **New Session Bean...**.
2. For **Name** and **Location**: Name the EJB as **CounterManufacturerEJB**.
3. Under **Package**, enter **beans**.
4. Mark **Session Type** as **Stateful**.
5. Leave **Create Interface** with nothing marked and click **Finish**.



Creating the business method

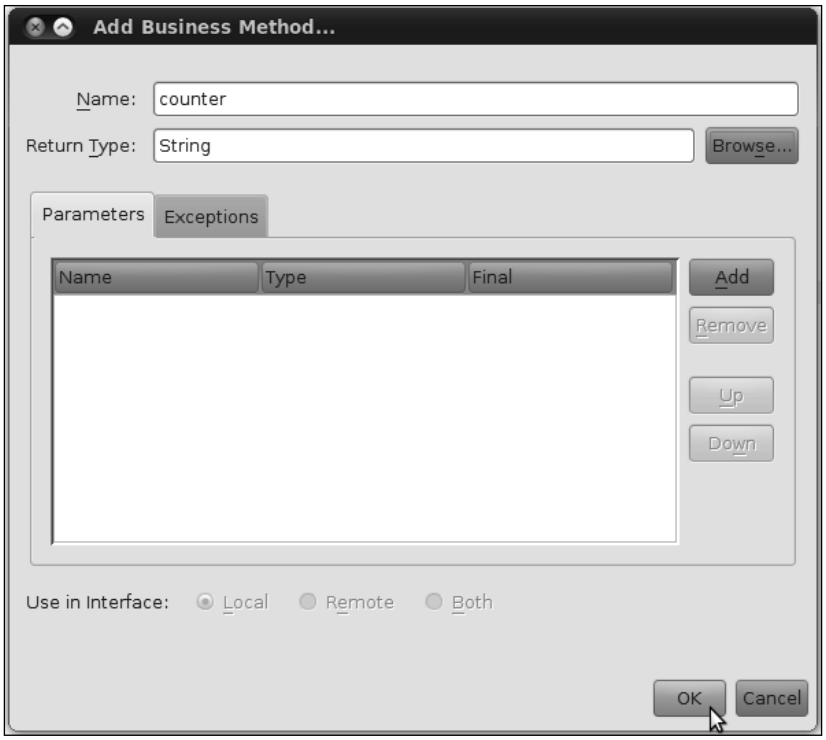
With **CounterManufacturerEJB** open, add the following variable:

```
private int counter = 0;
```

Then right-click inside the class body and select **Insert Code...** (or press *Alt+Insert*) and select **Add Business Method...**.

When the **Add Business Method...** window opens:

1. **Name** it as **counter** and for **Return Type**, enter **String**.
2. Click **OK.**



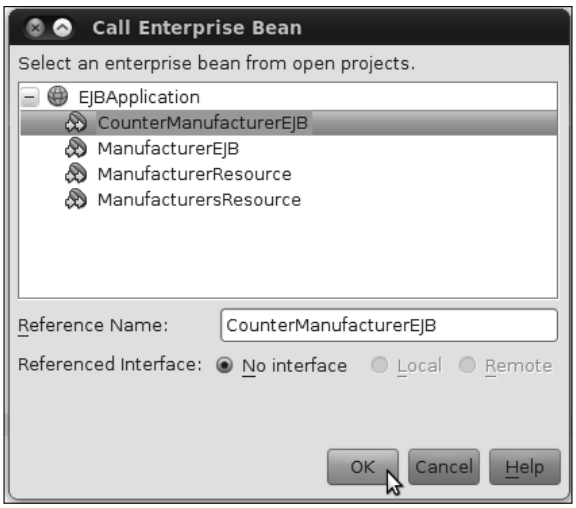
Replace the code inside the counter method with:

```
counter++;
return ""+counter;
```

Save the file.

Open `ManufacturerServlet` and after the class declaration and before the `processRequest` method:

1. Right-click and select **Insert Code...** or press *Alt+Insert*.
2. Select **Call Enterprise Bean...**.
3. In the **Call Enterprise Bean** window, expand the EJB Application node.
4. Select **CounterManufacturerEJB** and click **OK**.



Below we see how the bean is injected using annotation:

```
@EJB
CounterManufacturerEJB counterManufacturerEJB;
```

Resolve the import errors by pressing *Ctrl+Shift+I*.

Then add to the process request:

```
out.println("<b>Number of times counter was accessed<b>  " +
counterManufacturerEJB.counter() + "<br><br>" );
```

Save the file.

How it works...

NetBeans presents the user with a very easy-to-use wizard for creating beans. As with the stateless bean, we are presented with the different options for creating a bean.

This time we select the Stateful Bean. When clicking **Finish**, the IDE will create the EJB POJO class, place it in the beans package, and annotate, with `@Stateful`, the class signifying that we have created a Stateful Session Bean.

We then proceed to add the logic in our EJB. Through another wizard, NetBeans makes it easy to add a business method. After pressing *Alt+Insert*, we are presented with the choices of what can be done in that context. After adding the code, we are ready to integrate our EJB with the servlet.

Again, pressing *Alt+Insert* comes in handy when we want to create a reference to our EJB. After the correct bean is selected in the **Call Enterprise Bean** window, NetBeans creates the code:

```
CounterManufacturerEJB counterManufacturerEJB =
lookupCounterManufacturerEJBBean();
```

And also:

```
    private CounterManufacturerEJB lookupCounterManufacturerEJBBean()
{
        try {
            Context c = new InitialContext();
            return (CounterManufacturerEJB) c.lookup("java:global/
EJBApplication/CounterManufacturerEJB!beans.CounterManufacturerEJB");
        } catch (NamingException ne) {
            Logger.getLogger(getClass().getName()).log(Level.SEVERE,
"exception caught", ne);
            throw new RuntimeException(ne);
        }
    }
```

This boatload of code is created by the IDE and enables the developer to fine-tune things like logging over exceptions and other customizations. In fact, this is the way that EJB was called prior to annotations being introduced to Java EE. The method is simply calling the **application server context** with the lookup method, along with the **Remote Method Invocation** (**RMI**) naming conventions used to define our EJB and assign the reference to the object itself.

Notice that all this code could be simplified to:

```
@EJB
CounterManufacturerEJB counterManufacturerEJB;
```

But we tried to show how much liberty and options the developer has in NetBeans.

There's more...

Disabling GlassFish alive sessions.

GlassFish and sessions

To keep sessions alive in our Application Server GlassFish, we need to navigate to the Services window:

1. There we will need to expand the **Servers** node.
2. Right-click on **GlassFish** and select **Properties**.
3. Click on **Preserve Sessions Across Redeployment** if you do not want this feature.

This option preserves the HTTP sessions even when GlassFish has been redeployed. If the data has been stored in a session, it will be available next time a redeployment occurs.

Sharing a service through Web Service

Web services are APIs which, in the case of this recipe, access some data over a network from any platform and using any programming language.

In the world of cloud computing, web services have become an increasingly popular way for companies to let developers create applications using their data. A good example of this is Twitter. Thanks to exposition of Twitter data through web services, it has been possible to create numerous Twitter clients on virtually all platforms. In this recipe, we will create a web service that returns information from a database table; we will see that this information can be transferred either in XML or **JavaScript Object Notation** (**JSON**) format. JSON provides the user with data access simplicity, when compared to XML, since it does not need a bunch of tags and nested tags to work *Getting ready*

It is required to have NetBeans with Java EE support installed to continue with this recipe.

If this particular NetBeans version is not available in your machine, please visit:

`http://netbeans.org`

We will use the GlassFish Server in this recipe since it is the only server that supports Java EE 6 at the moment.

We also need to have Java DB configured. GlassFish already includes a copy of Java DB in its installation folder. If you wish to learn how to configure Java DB, refer to *Chapter 4*, *JDBC and NetBeans*.

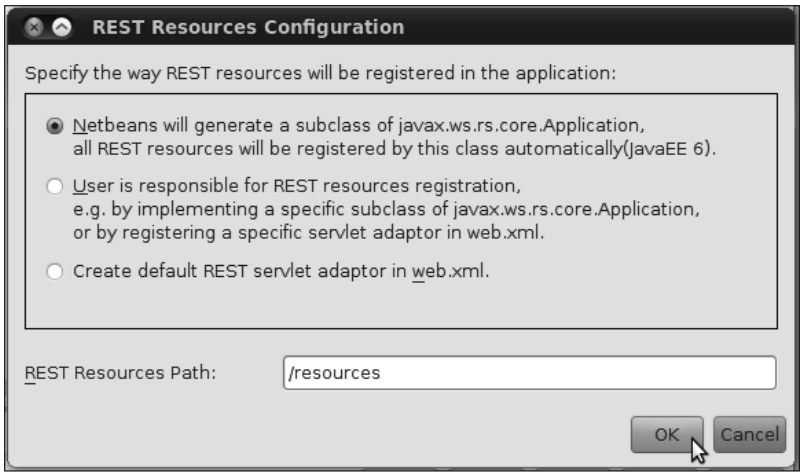
It is possible to create this recipe if an existing database schema and an EJB application exists. However, for the sake of brevity, we will use the sources from the previous recipes.

How to do it...

Right-click on the **EJBApplication** node, select **New** then **Other** then **Web Services and RESTFul Web Services from Entity Class**....

1. For **Entity Classes**: On **Available Entity classes**, select **Manufacturer**, click **Add**, and click **Next**.
2. For **Generated Classes**: Leave all the fields with their default values and click **Finish**.

A new dialog, **REST Resources Configuration**, pops-up; select the first option and click **OK**.



How it works...

The REST resources configuration asks the user which way the RESTful resources should be accessed, presenting the user with three different options. We have chosen to use `javax.ws.rs.core.Application` because it is the standard in Java EE 6 and, thus, increases the portability of the application, instead of the `web.xml` option. The second option allows the developer to code their way through registering the resources and choosing the service path.

To take a look at the generated files, expand the service package. Two java files are present: `AbstractFacade.java and ManufacturerFacadeREST.java`.

Opening the `ManufacturerFacadeREST.java` will show that this file is actually a stateless EJB created by the IDE that is used to interface with the database and retrieve information from it.

NetBeans also automatically generates a converter for our ManufacturerResource. This converter is used for creating a resource representation from the corresponding entity instance. Those classes can be found in the converter package.

There's more...

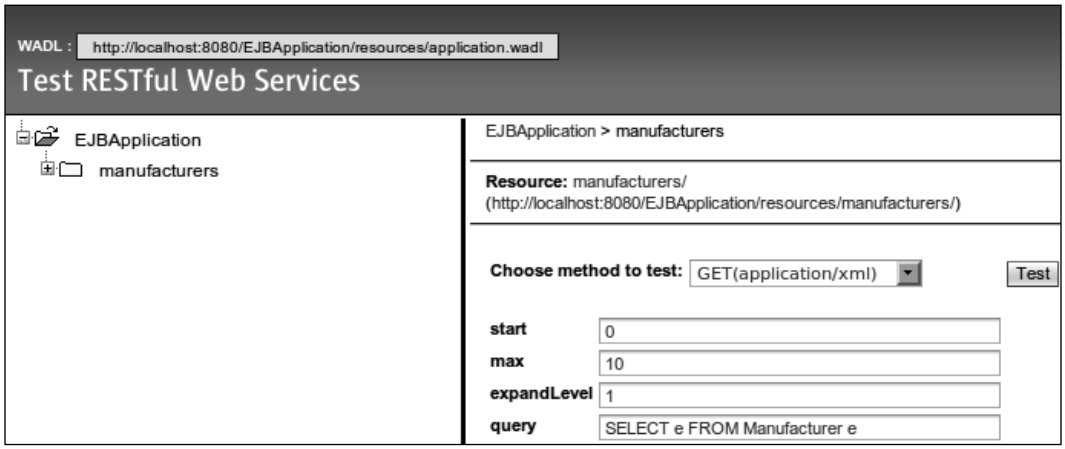
Using NetBeans to test the web services.

Testing the web service

Now that we have created a RESTful web service, we need to know if everything is working correctly or not.

To test our web service, right-click `EJBApplication` and select **Test RESTful Web Service**. NetBeans will be launched; deploy our application in GlassFish and then point the browser to the web service.

When the Test RESTful Web Service page opens, click on the **Test** button on the right side.



Upon clicking **Test**, the test request is sent to the server. The results can be seen in the response section.

Under **Tabular View**, it is possible to click in the URI and get the XML response from the server.

Raw View, on the other hand, returns the entire response, as it would be handled by an application.

It is also possible to change the format in which the response is generated. Simply click on the drop-down **Choose** method to test from **GET(application/xml)** to **GET(application/json)** and click **Test**. Then click on **Raw View** to get a glimpse of the response.

Creating a web service client

In this recipe, we will use Google Maps to show how NetBeans enables developers to quickly create an application using web services provided by third parties.

Getting ready

It is required to have NetBeans with Java EE support installed to continue with this recipe.

If this particular NetBeans version is not available in your machine, please visit:

```
http://netbeans.org
```

We will use the GlassFish Server in this recipe, since it is the only server that supports Java EE 6 at the moment.

For our recipe to work, we will need a valid key for the Google Maps API. The key can be found at:

```
http://code.google.com/apis/maps/signup.html
```

On the site, we will generate the key. Tick the box that says **I have read and agree with the terms and conditions**, after reading and agreeing of course.

Under **My website URL**, enter:

```
http://localhost:8080
```

Or the correct port in which GlassFish is registered.

Then click on **Generate API key**.

The generated key looks something like:

```
ABQIAFDAc4cEkV3R2yqZ_ooaRGXD1RT8M0brOpm-All5BF9Po1KBxRWWERQsusT9yyKEXQ
AGcYfTLTyArx88Uw
```

Save this key, we will be using it later.

How to do it...

Creating the Java Web Project

1. Click **File** and then **New Project** or Press *Ctrl+Shift+N*.
2. For **New Project**: On the **Categories** side, choose **Java Web** and on the **Projects side**, select **WebApplication**.
3. Click **Next**.
4. For **Name and Location**, under Project Name, enter **WebServiceClient**.
5. Tick the box on **Use Dedicated Folder for Storing Libraries**.
6. Now, either type the folder path or select one by clicking on **browse**.
7. After choosing the folder, we can proceed by clicking **Next**.
8. For **Server and Settings**: Under **Server**, choose **GlassFish Server 3.1**.
9. Leave the other options with their default values and click **Finish**.

Creating Servlet

Right-click on the **WebServiceClient** project, and select **New** and then **Servlet...**.

1. For **New Servlet**: Under **Class Name**, enter `WSClientServlet`.
2. And under **package**, enter **servlet**.
3. Click **Finish**.

When the WSClientServlet opens in the editor, remove the code starting with:

```
/* TODO output your page here
```

And ending with:

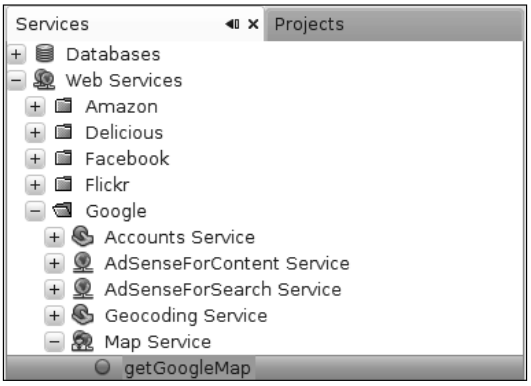
```
*/
```

And save the file.

Adding a Web Service

Navigate to the **Services** window and expand the **Web Services** node, followed by **Google**, and finally **Map Service**.

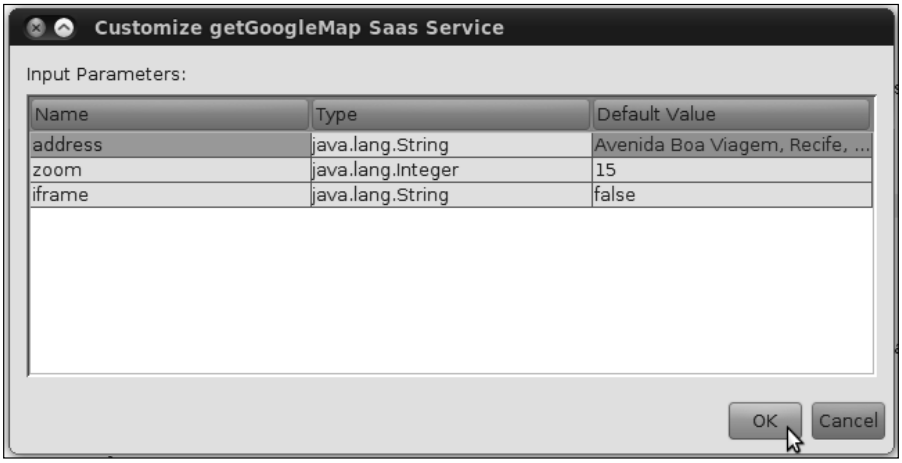
Accepting a security certificate is required to access this service and to continue with the recipe. Please refer the following screenshot:



Drag and drop `getGoogleMap` into our Servlets `processRequest` method.

A new window, **Customize getGoogleMap SaaS Service**, pops-up.

1. Under **Input Parameters**, double-click the cell on the **address** row under the **Default Value** column, to change the value to the desired address (or keep it default if the provided one is okay).
2. Click **OK**.



When the new block of code is written by NetBeans, uncomment the following line:

```
//out.println("The SaasService returned: "+result.getDataAsString());
```

Remember the key generated in the Getting Ready section?

In the Projects window, expand the **Source Packages** node and the package `org.netbeans.saas.google`, and double-click on `googlemapservice.properties`.

Paste the key after the **=** operator.

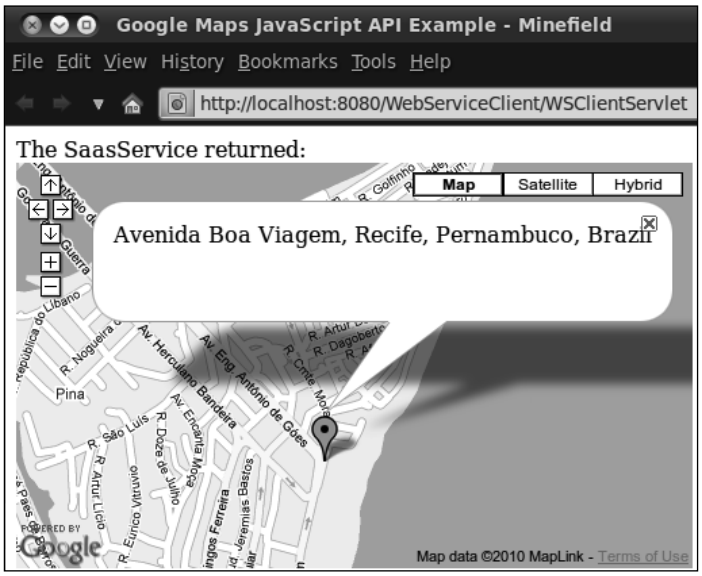
The line should look like:

```
api_key=ABQIAFDAc4cEkV3R2yqZ_ooaRGXD1RT8M0brOpm-All5BF9Po1KBxRWWERQsusT9yyKEXQAGcYfTLTyArx88Uw
```

Save file, open `WSClientServlet` and press *Shift+F6*.

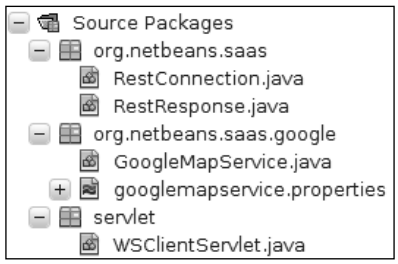
When the Set Servlet Execution URI window pops-up, click **OK**.

The browser will open with our application path already in place and it will display this:



How it works...

After dragging and dropping the Google Web Service to our class, a folder structure is created by NetBeans:



Let's check what is in our folder structure:

- **GoogleMapsService.java**: Responsible for checking the coordinates given by the developer, and checks and reads the key from the properties file.
  - Returns HTML text to access GoogleMap.
- **RestConnection.java**: Responsible for establishing the connection to the Google servers.
- **RestResponse.java**: Holds the actual data returned from Google.
- **GoogleMapsService**: The class that our Servlet uses to interact with the other classes and Google.

There's more...

Discovering other web services bundled with the IDE.

Other services

There are many other web services available in the Web Service section of the IDE.

Services such as:

- Amazon: EC2 and S3
- Flickr
- WeatherBug

It is just a matter of checking the documentation of the service provider, and starting to code your own implementation. Try it out!

Where to buy this book

You can buy NetBeans IDE 7 Cookbook from the Packt Publishing website:
`http://www.packtpub.com/netbeans-ide-7-cookbook/book`

Free shipping to the US, UK, Europe and selected Asian countries. For more information, please read our shipping policy.

Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.

www.PacktPub.com