NetBeans IDE Field Guide

Table of Contents

Extending Web Applications with Business Logic: Introducing EJB Components

For many NetBeans users, as well as Web application developers, the Enterprise JavaBeans concept might be new or apparently complex, as this is the first time that NetBeans 4.1 exposes the necessary wizards and features to easily create EJB components, and add business methods to them. Once these business methods are implemented (in Java code), then they can be called either from other EJB components or from a Web application's servlets or utility classes. The benefits of encapsulating application code within EJB business method are numerous:

1. Enterprise beans support transactions, the mechanisms that manage the concurrent access of shared objects. Transaction setting are declarative, via the deployment descriptor files.
2. Enterprise beans can be used by many clients, across machines or not (remote and/or local access)
3. Enterprise beans business methods can be secured declaratively, without source code modification.
4. Enterprise beans access external resources like databases, message queues, mail sessions, Web services, declaratively via JNDI naming: the Java Naming and Directory Interface (JNDI) naming service enables components to locate other components and resources. To locate a JDBC resource, for example, an enterprise bean invokes the JNDI `lookup` method. The JNDI naming service

maintains a set of bindings that relate names to objects. The `lookup` method passes a JNDI name parameter and returns the related object.

See Table 8-1 for a list of all of the EJB types.

Table 8-1: Types of Enterprise Beans

Enterprise Bean type	Description
Session	Performs a task for a client; implements a Web service. A Session Bean can be stateful for conversation handling between the client (the user of the business logic) and the Server, or stateless.
Entity	Represents a business entity object that exists in persistent storage, typically SQL databases (and possibly others)
Message-Driven	Acts as a listener for the Java Message Service API, processing messages asynchronously

EJB Project type Wizards.

The first thing to do in order to develop Enterprise JavaBeans components is to create an EJB Module project that can contain one or more EJB components. To give an analogy from the Web application concept: a Web application is a deployable J2EE component containing a collections of servlets, web pages and JSP files, while an EJB module is also a deployable J2EE component, containing a collection of enterprise beans.

To create an EJB Module project:

1. Choose the EJB Module project type, which is in the Enterprise project categories (as shown in Figure 8-1).

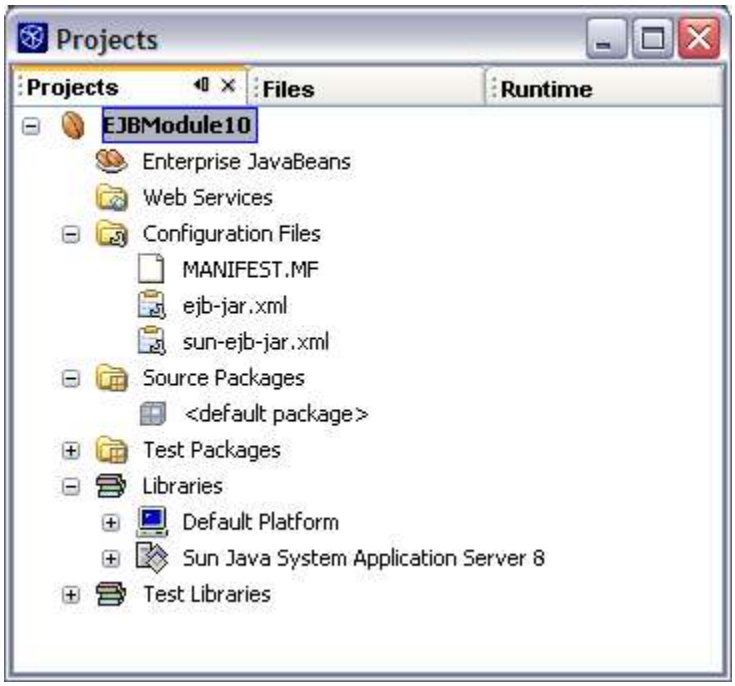
Figure 8-1
New Project Wizard with EJB Module project type selected

2. Specify the location of the project, its name, and whether or not you want to add this J2EE component to an existing J2EE application (EAR project).

   You can add the module a J2EE Application project later – for example, when you create the J2EE project.

Once you complete the wizard, your project is created and visible in the Projects window as shown in Figure 8-2. (For now, it contains no Enterprise JavaBeans components. Adding EJB Components, Files and Libraries to Your EJB Module on page XXX explains how to populate the module.)

Figure 8-2



Projects window with EJB module project showing

You can select the Files window (as shown in Figure 8-3) to see which directories and files have been created on disk. See EJB Module Structure below for a description of the conventions used for this structure.

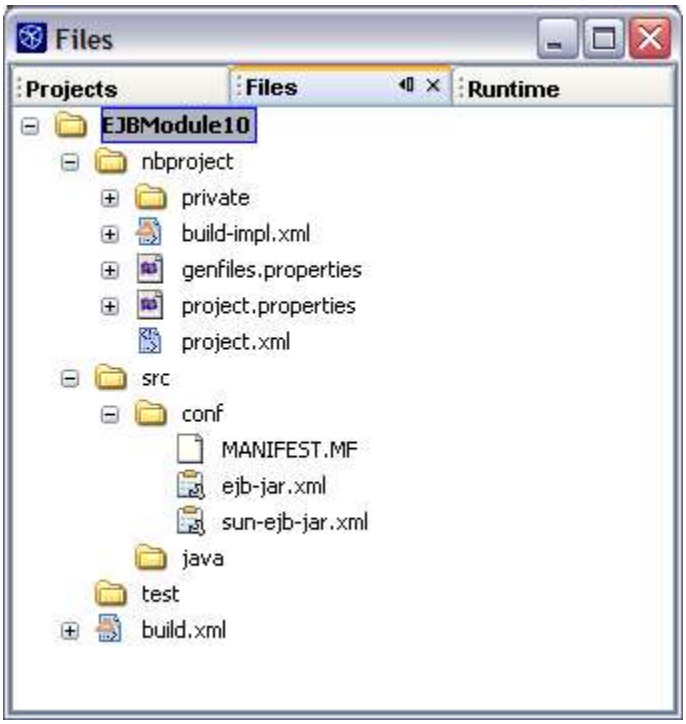


Figure 8-3
Files window with EJB module project showing

EJB Module Structure

The J2EE Blueprints provide guidelines on how to structure your J2EE applications and EJB Modules to ensure that they work properly with different application servers. When you create a project in the IDE, this J2EE Blueprints convention structure is respected.

The following is a quick description on the structural elements of the built EJB module:

- The `src/java` folder which contains all the Java source files in the application.
- The `src/conf` folder which contains the J2EE deployment descriptors and the application servers specific deployment descriptors.
- The `setup` directory which contains server specific resource files.

You can find additional information on these conventions at

https://conventions.dev.java.net/

See Table 8-2 for information on how the various source elements of an EJB Module map to their representation in the IDE and where they end up in the deployed component.

Table 8-2: Matrix of EJB Module Elements and Their Representation in the IDE

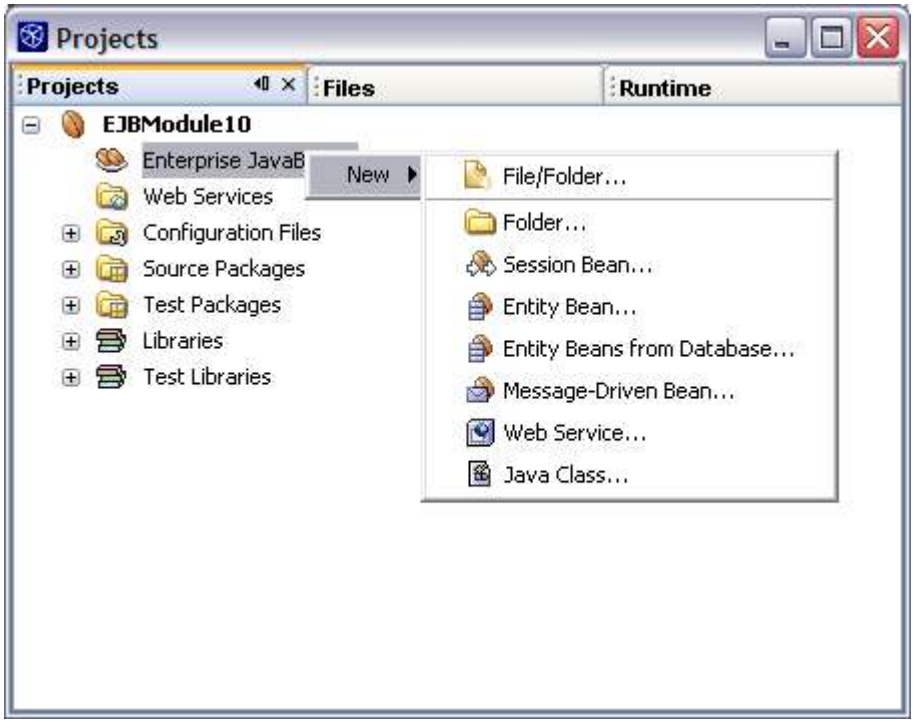
Content	**Representation in the Projects Window**	**Representation in the Files Window**	**Location Within the Built EJB JAR File (located in the dist folder)**
EJBs	Enterprise JavaBeans node	`src/java`folder	root of the file
Java source files, helper classes, EJB Java files, etc.	Source Packages node	`src/java` folder	Package structure for the JAR file
unit tests	Test Packages node	`test` folder	N/A
deployment descriptor (ejb-jar`.xml`, `webservices.xml`)	Configuration Files node	`src/conf` folder	`META-INF` folder
Application servers deployment descriptors context configuration file (`sun-ejb-jar.xml`, `sun-cmp-mapping.xml`, `others`)		`src/conf` folder	`META-INF` folder
Application Server specific resources or scripts (SQL,...)	Configuration Files (visible when some J2EE resources exist there)	`setup` folder	N/A. The resources in this folder are registered automatically at deployment time for the Sun Application Server target.
Web services	Web services node	src area (Java code)	
libraries	Libraries node	Location of the libraries folder	JAR libraries included in the EJB module JAR file, at the top location.
Test classpath entries	Test Libraries node	`test` folder	N/A
project metadata including build script	Project Properties dialog box, which you can open by right-clicking the project's node and choosing Properties.	build.xml file, nbproject folder	N/A
EJB Module build area (*.class files)	Not visible in the project view	Build and build/generated	Main content for the EJB module archive jar file.

Adding EJB Components, Files and Libraries to Your EJB Module

Once you have created an EJB module project through the New Project wizard, you can start populating it with new EJB components and helper Java classes.

The most straightforward way to create files is by opening the Projects window, right-clicking the node where you want to place the file and choosing New and then a template from the submenu. A short wizard appears for the template enabling you to set the name and other characteristics of the file. Choose the “Session Bean” template as shown in Figure 8-4.

Figure 8-4



Adding a File to an EJB Module

The wizard will create a session bean and add it to the EJB Module. You can specify the bean name, package (make sure you select or create a Java package there instead of leaving it blank). You can specify whether this session bean will have a local and/or a remote interface (local is the default), and if the bean is stateful or stateless. For now, accept the default values to create a local stateless session bean as shown in Figure 8-5.

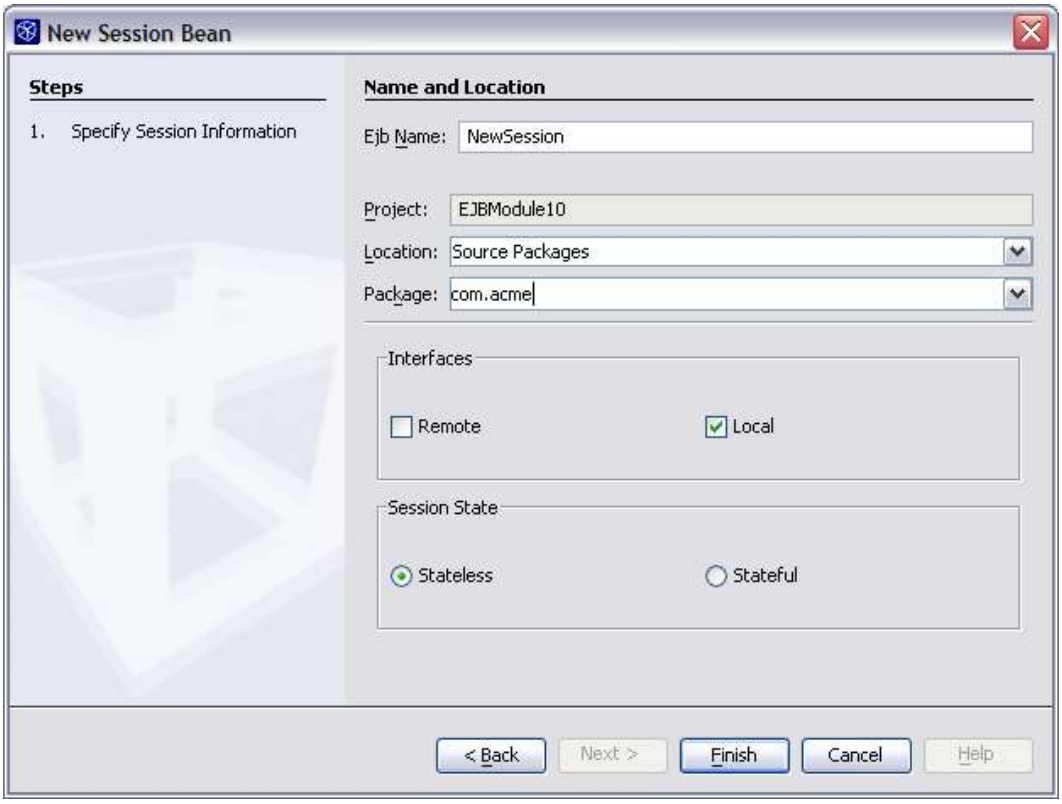


Figure 8-5
New Session Bean page of the New File wizard

Notice the new logical view in the Projects window under the Enterprise JavaBeans node (as shown in Figure 8-6). This EJB component is a set of Java files (4 for a simple session bean):

- The local interface (*BeanName*`local.java`)
- The local home interface (*BeanName*`LocalHome.java`)
- The bean implementation itself (*BeanName*`Bean.java`)
- The business interface (*BeanName*`LocalBusiness.java`)

The Project window's logicall view hides the complexity of this enterprise bean by showing only a single node that exposes some important methods for the bean, like the local methods.

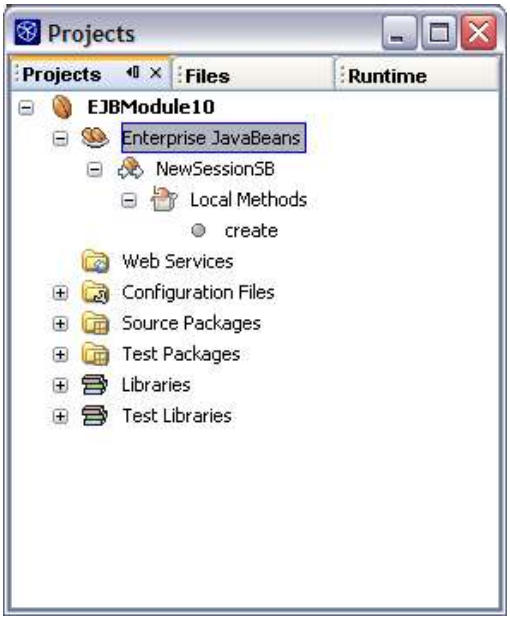


Figure 8-6
Projects window showing a new EJB component.

> **NetBeans IDE Tip**
>
> "Where are my Java Files?" The Projects window hides the Java files by default. But you can see the entire package structure and all the Enterprise Beans classes under the Source Packages node. Remember that an EJB component is a collection of Java files, and some entries in some deployment descriptors files. NetBeans will keep all these files synchronized automatically whenever you interact with the Projects window.

Adding Business Logic to an Enterprise JavaBeans Component

In this section you will learn about the different method types you can add to an Enterprise JavaBeans component. Remember that NetBeans IDE offers the necessary wizards that will greatly simplify the work of coding business logic within EJB components.

Bean Methods

A J2EE application gets its work done through methods that the client calls on the bean. The following list of method types are either automatically generated via the wizard with default implementation, or can be added via popup menu actions in the Projects window or the Source Editor window for an EJB bean implementation class. Necessary extra generation is taken care of by the IDE (i.e. updating the local or remote interfaces with the correct signature entry):

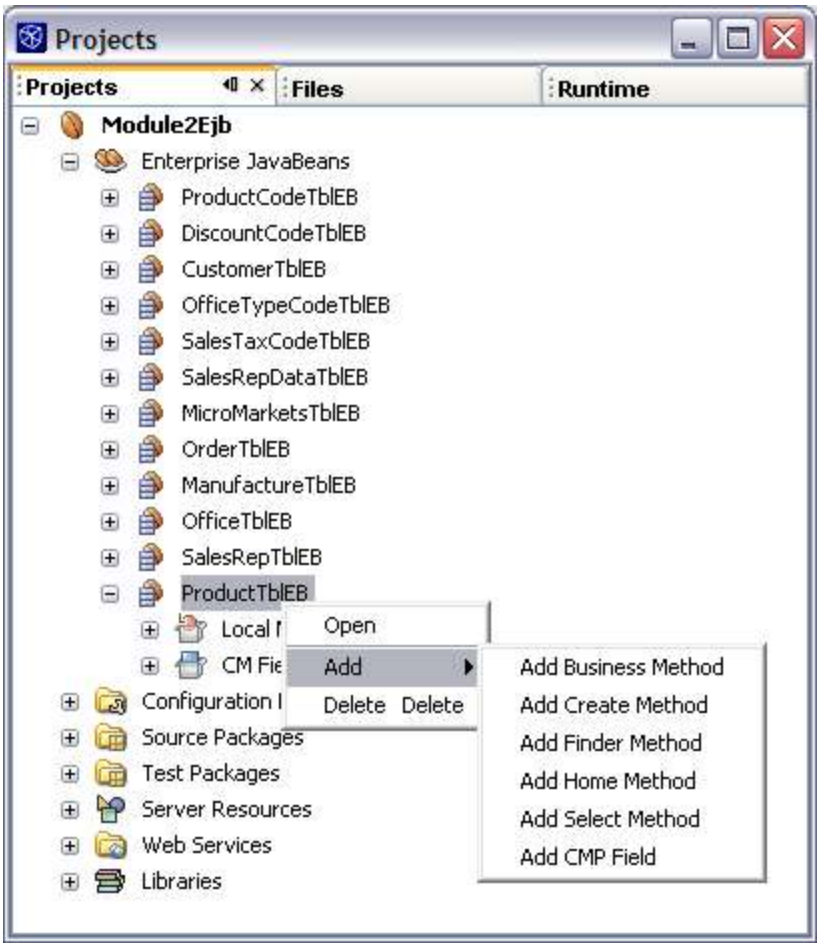


Figure 8-7
Projects window and the adding of a method to an EJB component

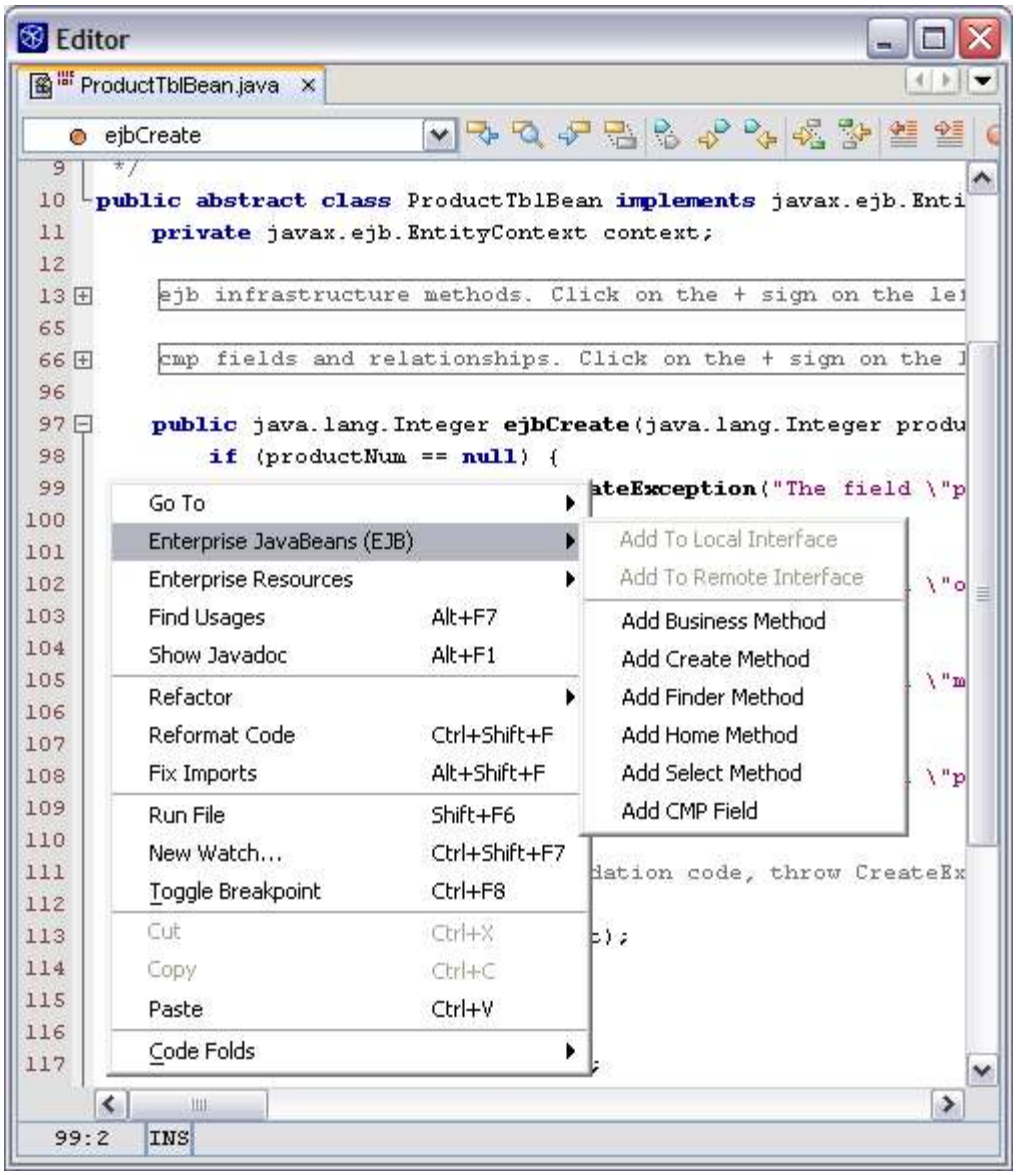


Figure 8-8
Source Editor and the adding of a method to an EJB component

- **Business Methods.** A client calls business methods on a bean through the bean's remote interface (or local interface, as applicable).

  You have to add business methods to the bean yourself – the IDE doesn't generate any default business method declarations. However, when you specify a business method, the IDE places matching method declarations in the bean class and in the remote, local, or remote and local interfaces.

  NetBeans IDE Tip

  Business Methods are the most important methods for an EJB component. These are the ones called by other EJBs or Web tier components like JSP files or servlets. A special NetBeans wizard is available to simplify the coding of calling an EJB business method.

From a servlet or EJB file in the Source Editor, right-click to activate the popup menu and choose the “Enterprise Resources” menu and the Call EJB sub-menu item.

- **Life-cycle Methods.** The container calls several methods to manage the life cycle of an enterprise bean. Depending on the type of bean, the container works through the methods in slightly different ways. You have the option of specifying parameters for some of these methods.

  The IDE automatically generates the appropriate life-cycle method declarations for each type of bean and places them in the bean class.
- **Finder Methods.** The client goes through the home interface to find an entity bean instance by its primary key. The developer can also add other finder methods.

  NetBeans automatically generates a `findByPrimaryKey` method declaration in the local home interface of every entity bean (and in the bean's home interface, if it has one). The IDE also places a corresponding `ejbFindByPrimaryKey` method declaration in the bean class of every entity bean that manages its own persistence (that is, a bean-managed persistent entity bean, or BMP entity bean). If you add another finder method, the IDE automatically places the corresponding method declarations in the local home (and home) interface and, for BMP entity beans, in the bean class.

  An entity bean that delegates its persistence to the container is called a container-managed persistent entity bean, or a CMP entity bean. Finder methods that are added to CMP entity beans include EJB Query Language (EJB QL) statements, which are converted automatically to the kind of SQL code the server needs.

  NetBeans IDE Tip

  The server integration plug-in module is what does this EJBQL conversion. NetBeans IDE comes with a server integration plug-in for the Sun Java System Application Server. If you deploy to a different application server, you would need a corresponding plug-in module for that server to get the same functionality in the IDE as you get when deploying to the Sun Java System Application Server.

- **Create Methods.** The container initializes the enterprise bean instance, using the create method's arguments.
- **Home Methods.** An entity bean can use a home method for a lightweight operation that doesn't require access to any particular instance of the bean. (By contrast, a business method does require access to a particular instance.) It is up to you to explicitly add the home method, and the IDE generates the corresponding method declaration in the bean class and the bean's local home or home interface. An entity bean can have any number of home methods.
- **Select Methods.** A CMP entity bean can use a select method. Like a finder method, a select method can query the database and return a local or remote interface or a collection. In addition, a select method can query a related entity bean within the same EJB module and return values from its persistent fields. Select methods aren't exposed in remote-type interfaces and can't be invoked by a client.
- `OnMessage` **Methods.** A client sends a message through a Java Message Service (JMS) destination to call an `onMessage` method on a message-

driven bean.

Adding a Simple Business Method

To add a business method:

1. Choose the Add Business Method popup menu item either from an EJB component's node (as shown in Figure 8-7) or from within the Source Editor for a bean implementation class (as shown in Figure 8-8).
2. In the Add Business Method dialog box (shown in Figure 8-9), enter a method name, a list of parameters and their type, as well as possible exceptions that will be thrown by this business method.

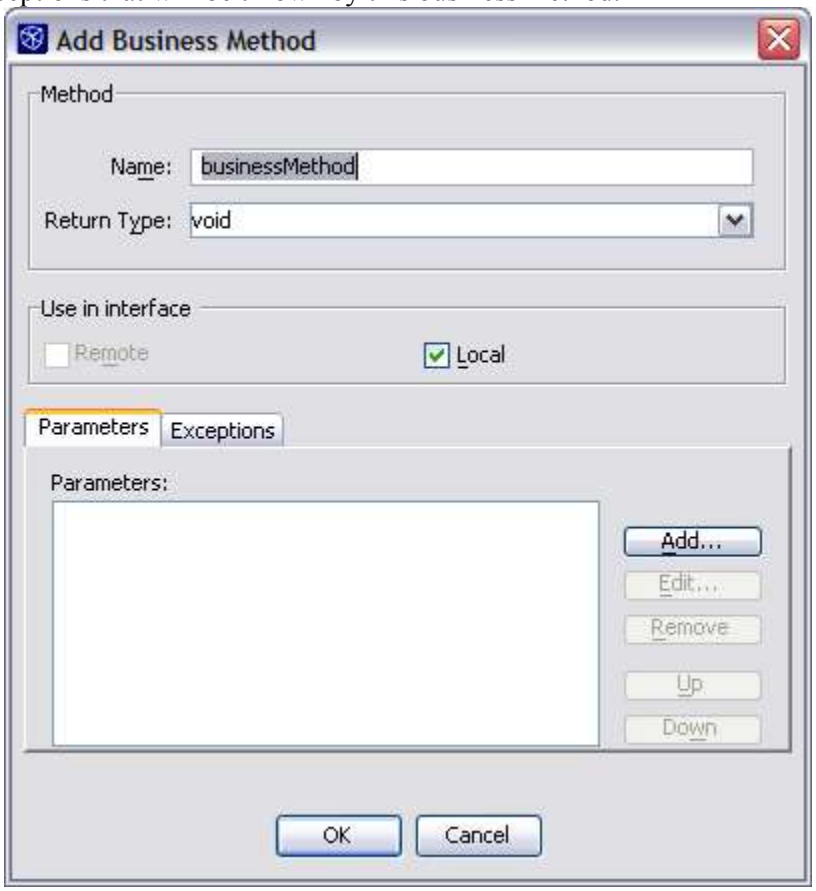
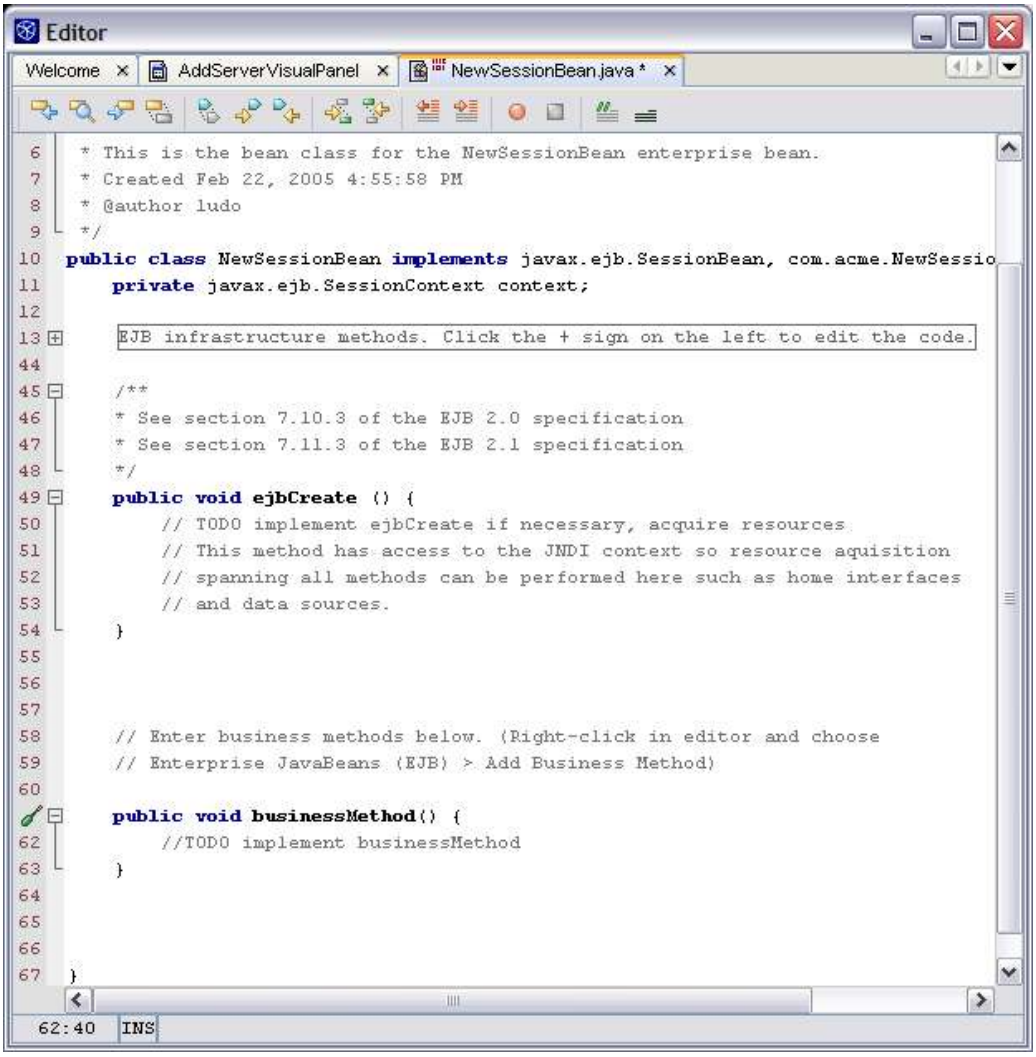


Figure 8-9
Add Business Method dialog box

Notice the change inside the bean implementation Java file (as shown in Figure 8-10). It now contains the new business method body and you now can use the capabilities of the IDE's Source Editor to finish the implementation of the method.

Figure 8-10



Source Editor with new business method added

3. When you are done coding the method, right-click your project's node in the Projects window and choose Build Project to trigger the compilation of the Java files and the creation of the EJB Archive file (in the `dist` directory).

EJB Bean Deployment Descriptors

One of the goals of NetBeans IDE is to keep you from having to deal with deployment descriptors as much as possible. This is achieved via the zero-configuration concept implemented in the IDE: when you use the provided commands, such as Call EJB, Use Database, Call Message, or Call Web Service Operation, the IDE performs the following tasks:

- Generates the Java Code snippet for the JDNI lookup code in the caller Java file.
- Makes sure to update the J2EE deployment descriptor file by adding the

corresponding EJB-REF or RESOURCE-REF elements.

- Modifies the J2EE project to add the necessary project dependancies if the call goes to a class or resource in another IDE project. The resulting packaging has to use the J2EE Application Project type to make sure all the J2EE modules are correctly assembled into a J2EE Application Archive (EAR) before deployment.

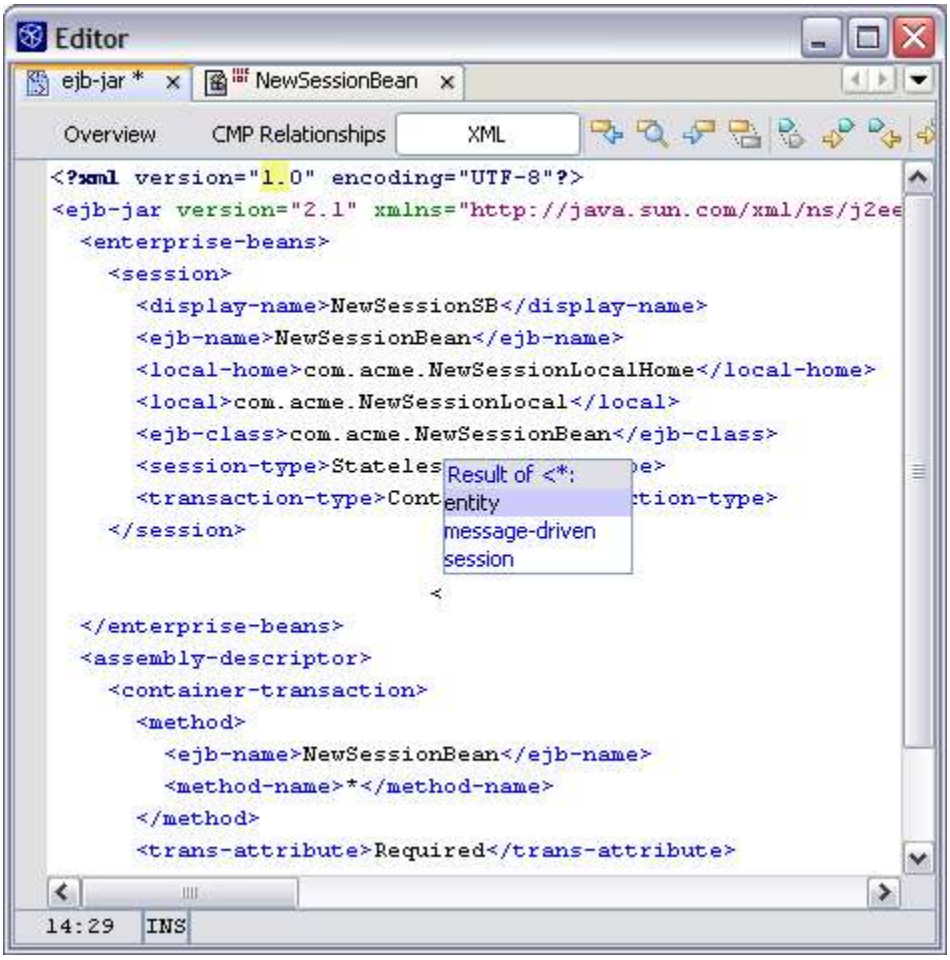
Of course, you are still allowed to edit the deployment descriptors. And here also, NetBeans offers significant ease of use features like two editing modes (direct XML, editing with code completion and online validation features, and visual editing).

The visual editor for a Deployment descriptor file (as shown in Figure 8-11) is activated by double clicking on the deployment descriptor's file node in the Projects window. A set of views are available: for example, for a J2EE EJB module, Overview and XML views are available. Also, a combo-box to the right of the buttons for the views allows you to jump directly to a particular section in the deployment descriptor file.

Figure 8-11
Visual deployment descriptor editor

If you want to edit the file's XML code directly, you can switch to the XML view of the file by clicking the XML button located at the top of the visual editor.

Figure 8-12



XML deployment descriptor editor with code completion

The XML editor (as shown in Figure 8-12) also has a code completion feature to make hand coding faster and more accurate. See Chapter 4, Generating Code Snippets for a general discussion of how code completion works.

Also note the toolbar on this editor. The last icon activates the XML validation action, so that you can verify the conformance of the deployment descriptor file with the DTD or schema. See Figure 8-13 for an example of the output after running the XML Validation command.

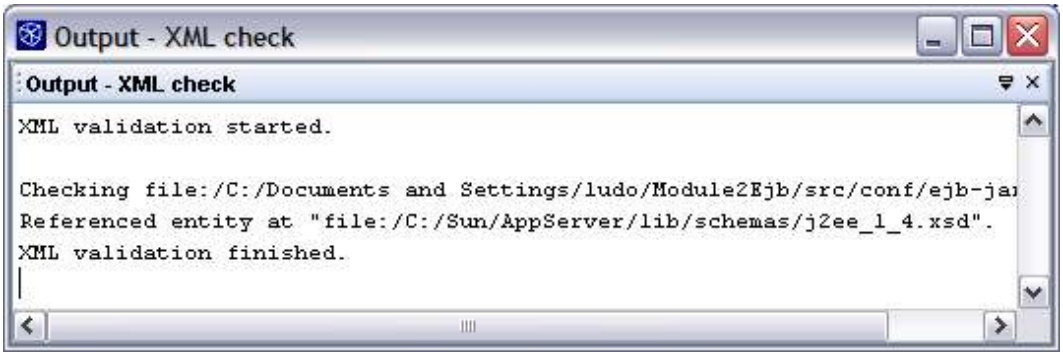


Figure 8-13
Output of XML validation