Borland®

Getting Started Guide – part I

Creating your first Enterprise JavaBean

Stateless Session Bean

by Gerard van der Pol and Michael Faisst, Inprise BV

Table of Contents

Preface

This document provides an overview of the development process of an Enterprise JavaBean (EJB)™ with **Borland® JBuilder™ 4 Enterprise Edition** and **Inprise® Application Server™ 4.1.**

Note
A developers license of the Inprise Application Server is shipped with JBuilder.

It is not a reference on developing Enterprise JavaBeans™; instead it is designed to get you jump-started using both products.

Definitions

Acronym	Description
CORBA®	Common Object Request Broker Architecture
EJB™	Enterprise JavaBeans™
RMI	Remote Method Invocation
JPDA	Java Platform Debugger Architecture

JBuilder™

white paper

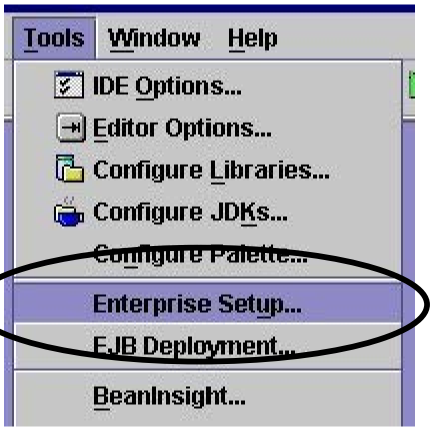
Additional information

Inprise Application Server (now Borland® AppServer™)
http://www.borland.com/appserver/
Borland JBuilder
http://www.borland.com/jbuilder/
Borland AppCenter
http://www.borland.com/appcenter/
Borland Enterprise JavaBeans Programmers Guide
http://www.borland.com/techpubs/appserver/
Enterprise JavaBeans with Paper
http://www.borland.com/visibroker/ ,click on White Papers
Sun's Java® 2 Platform Enterprise Edition
http://java.sun.com/j2ee/
Sun's Enterprise JavaBeans
http://java.sun.com/products/ejb/

Configuring JBuilder™ for EJB development

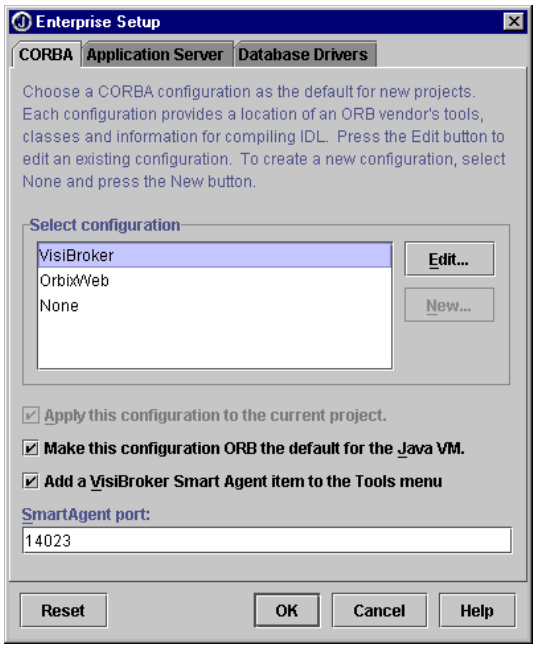
Before we start developing our first Enterprise JavaBean, let's spend some time on what's needed to get started. We assume you have installed both products on your machine. For more information on the installation refer to the Installation Guide.

After installation you need to configure JBuilder in order to enable Enterprise JavaBean Development. We will take a look at the Enterprise Setup, Default Project Properties, and configuring your libraries. Select **Tools, Enterprise Setup…** to open the Enterprise Setup dialog.



The Enterprise Setup Dialog enables you to configure JBuilder for CORBA® and EJB development and also add your JDBC® drivers to the JBuilder Environment.

The first thing to do is set up CORBA. If you don't select a CORBA configuration other than None, you will not be able to use the CORBA Wizards on the Enterprise tab of the Object Gallery.

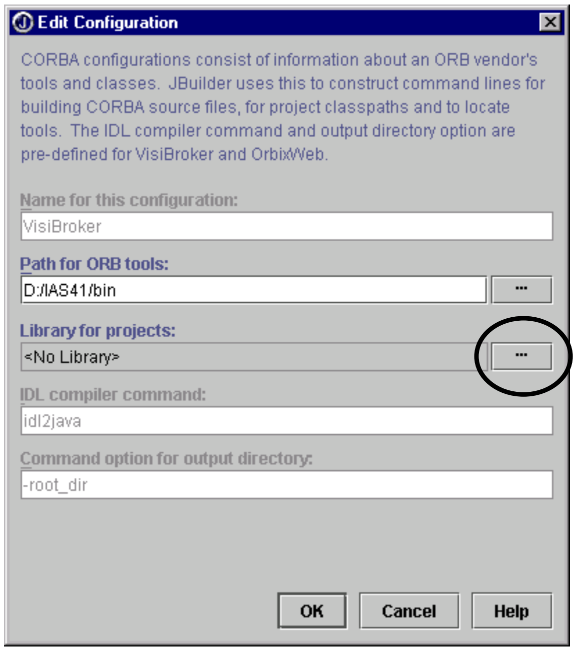


Choose a CORBA configuration as the default for new projects. Each configuration provides a location of an ORB vendor's tools, classes and information for compiling IDL. Press the Edit button to edit an existing configuration. To create a new configuration, select None and press the New button.

Select configuration
VisiBroker
OrbixWeb
None

☑ Apply this configuration to the current project.
☑ Make this configuration ORB the default for the Java VM.
☑ Add a VisiBroker Smart Agent item to the Tools menu
SmartAgent port: 14023

JBuilder ships with Inprise Application Server 4.1, which is based upon VisiBroker® for Java 4.1, so select VisiBroker from the list of Object Request Brokers. To complete the CORBA configuration, click on **Edit.** The Edit Configuration dialog appears.



CORBA configurations consist of information about an ORB vendor's tools and classes. JBuilder uses this to construct command lines for building CORBA source files, for project classpaths and to locate tools. The IDL compiler command and output directory option are pre-defined for VisiBroker and OrbixWeb.

Name for this configuration: VisiBroker

Path for ORB tools: D:/IAS41/bin

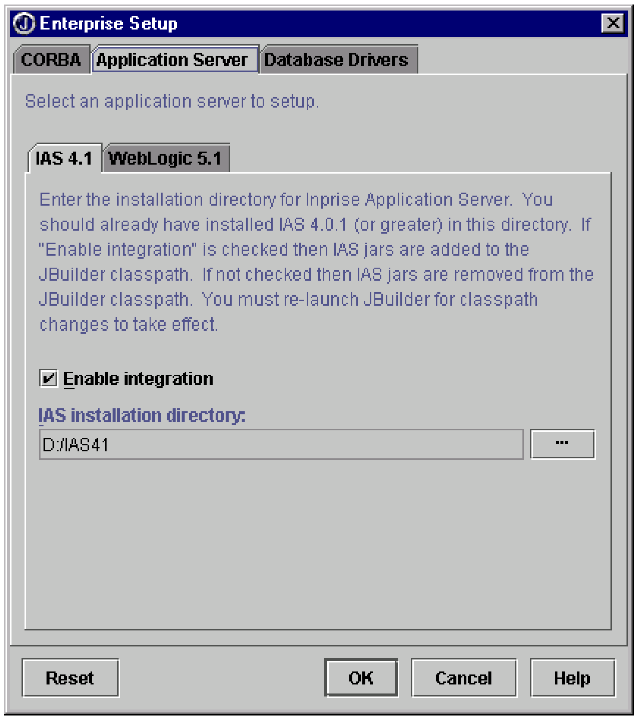
Library for projects: <No Library>

IDL compiler command: idl2java

Command option for output directory: -root_dir

First, enter the path for the ORB® Tools. Select the path where the VisiBroker tools, such as the IDL2JAVA compiler, reside. In this case, we select the BIN folder of the directory where you installed the Inprise Application Server.

Next is configuring the Application Server. Select the Application Server tab. You will have to specify the installation details of your Application Server to enable the integration with your Application Server and JBuilder. This integration gives you full control over the development and deployment process.



Select an application server to setup.

Enter the installation directory for Inprise Application Server. You should already have installed IAS 4.0.1 (or greater) in this directory. If "Enable integration" is checked then IAS jars are added to the JBuilder classpath. If not checked then IAS jars are removed from the JBuilder classpath. You must re-launch JBuilder for classpath changes to take effect.

[✓] Enable integration

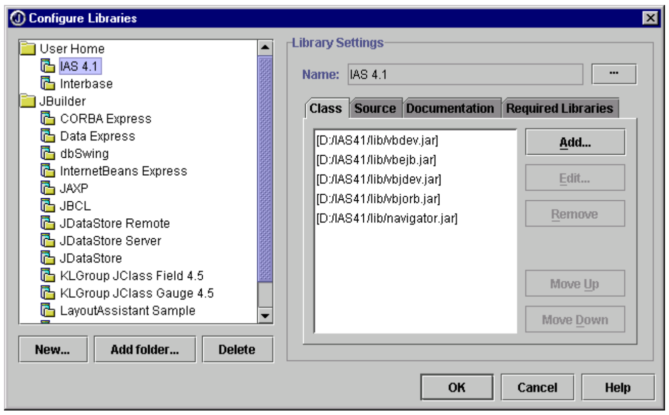
IAS installation directory: D:/IAS41

To enable integration, select the checkbox and specify the installation path of IAS. Those who have set this up under earlier versions of JBuilder will find that integrating Inprise Application Server in to JBuilder is made even simpler than it already was. JBuilder 4 has a many new EJB features that makes developing, debugging, and deploying your Enterprise JavaBeans, very easy. For development, JBuilder has a number of wizards to help you code your beans quickly; for deployment, there is the new build process, the Deployment Wizard, and Deployment Descriptor Editor. This last wizard gives you access to the XML deployment descriptor from within JBuilder and also adds the ejb-inprise.xml to the project. The EJB Deployment Wizard makes it possible to deploy the developed beans to the Application Server directly. Later on in this document there will be further explanations on the use of these wizards.

The integration of these tools/wizards is based on the Open tools API of JBuilder. This feature is well documented within JBuilder documentation. To start building your own Open Is everything *tools add-on, look at the website of Blake Stone found at: http://homepages.borland.com/bstone/index.html*

Configuring your Libraries.

To have complete control over your libraries, select **Tools,Configure Libraries…**



User Home
- IAS 4.1
- Interbase

JBuilder
- CORBA Express
- Data Express
- dbSwing
- InternetBeans Express
- JAXP
- JBCL
- JDataStore Remote
- JDataStore Server
- JDataStore
- KLGroup JClass Field 4.5
- KLGroup JClass Gauge 4.5
- LayoutAssistant Sample

Name: IAS 4.1

```
[D:/IAS41/lib/vbdev.jar]
[D:/IAS41/lib/vbejb.jar]
[D:/IAS41/lib/vbjdev.jar]
[D:/IAS41/lib/vbjorb.jar]
[D:/IAS41/lib/navigator.jar]
```

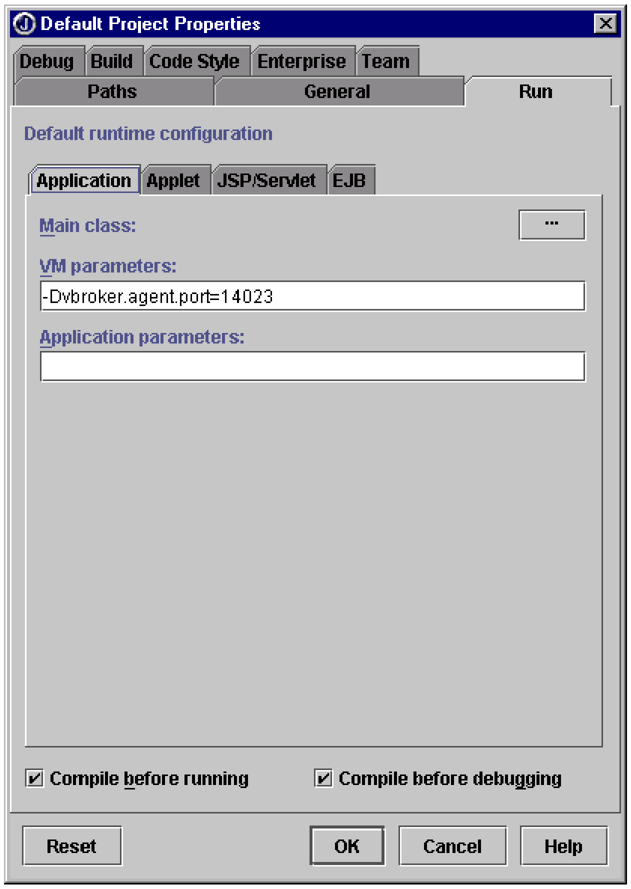
Configure Libraries allows you to add the path to the source (if you have source available), documentation, and required libraries of your library.

In the above figure you see the IAS 4.1 library we just created. You might notice that the number of jar files listed are not the same as we added before. In the case of our IAS 4.1 Library, JBuilder will automatically filter out the jar files it needs for the integration with Inprise Application Server. Later we will see where JBuilder get the info to do this.

Default Project Properties

Next on our Configuration Tour is a stop at the Default Project Properties dialog. Default project settings are stored in an actual project called Default.jpr found in the /.jbuilder 4 subdirectory of your Home directory. This Default.jpr file is used as a template whenever a new project is created. To change the default project properties, choose **Project|Default Project Properties** from the main menu.

JBuilder4 adds a number of properties to configure your project. See the online documentation for a complete description of this feature. For now we will only focus on the Run and Enterprise Properties.



Main class:

VM parameters: -Dvbroker.agent.port=14023

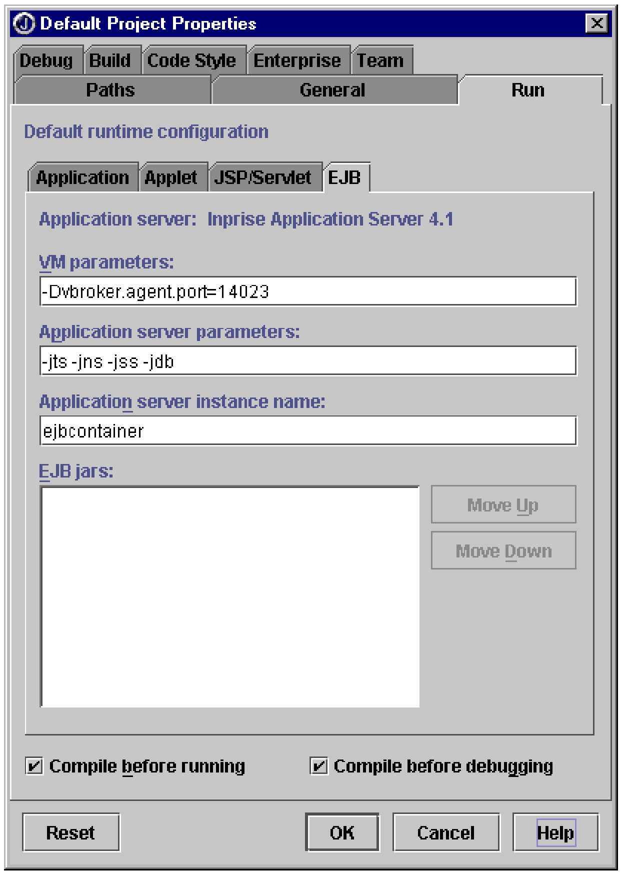
Application parameters:

[✓] Compile before running [✓] Compile before debugging

On the run tab you can specify the parameters to pass to your application, EJB Container or Virtual Machine. On the Application Tab, specify the Smart Agent Port you configured at the CORBA setup.

The EJB Run Properties allow you to specify the services to be started by the EJB Container, the default instance name of the Container and also the Smart Agent Port to use.

You can also specify the EBJ Jarfiles that the Container has to load on startup. We will see later how this feature is used.



Default runtime configuration

Application server: Inprise Application Server 4.1

VM parameters: -Dvbroker.agent.port=14023

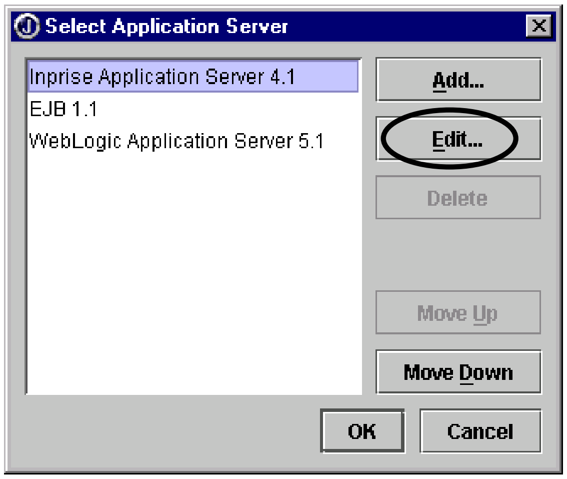
Application server parameters: -jts -jns -jss -jdb

Application server instance name: ejbcontainer

EJB jars:

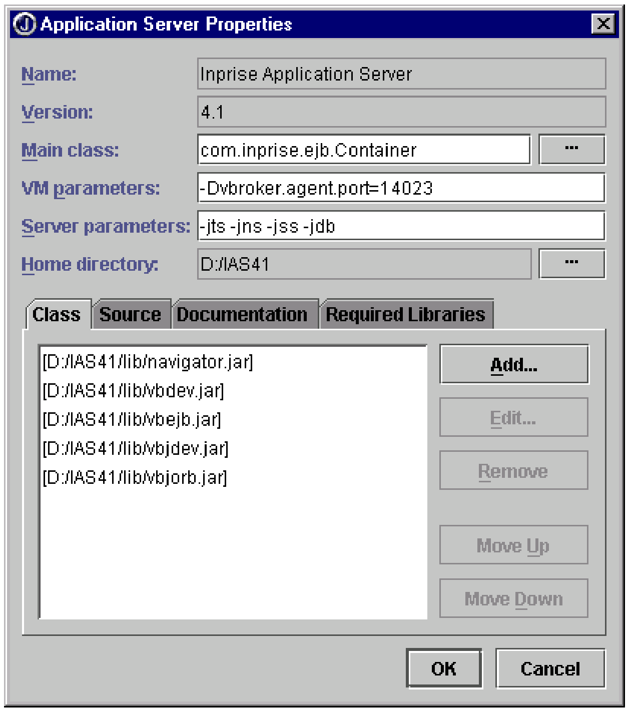
☑ Compile before running ☑ Compile before debugging

Next, we click on the Enterprise Tab of the Default Project properties dialog. On this page we can configure the Application Server. Click on the **...** button to open the Select Application Server dialog.



Inprise Application Server 4.1
EJB 1.1
WebLogic Application Server 5.1

Click on **Edit** to open the Application Server Properties dialog. This dialog enables us to configure the main class of the Application Server, VM parameters, Server parameters, and the libraries belonging to IAS. This information is already available after you install JBuilder. This is the place where JBuilder collects the information to filter the libraries in the CORBA setup.



Name: Inprise Application Server

Version: 4.1

Main class: com.inprise.ejb.Container

VM parameters: -Dvbroker.agent.port=14023

Server parameters: -jts -jns -jss -jdb

Home directory: D:/IAS41

[D:/IAS41/lib/navigator.jar]
[D:/IAS41/lib/vbdev.jar]
[D:/IAS41/lib/vbejb.jar]
[D:/IAS41/lib/vbjdev.jar]
[D:/IAS41/lib/vbjorb.jar]

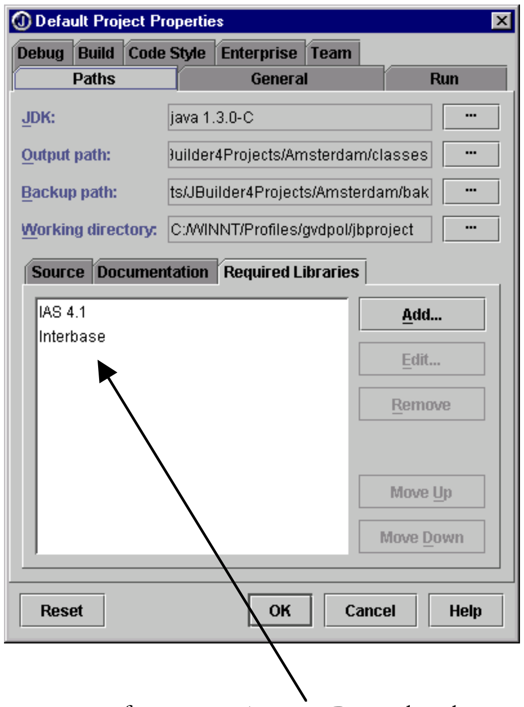
The Server parameters in the case of the Borland EJBContainer are:

- Jts for starting the Leightweight Transaction Service;
- Jns for starting the Java Naming Service;

- Jss for starting the Java Storage Service; and
- Jdb for starting the Java Database.

Project properties.

You can set the properties for your current project by right-clicking a .jpr or .jpx project file in the project pane and selecting Properties or by choosing **Project|Project Properties**.



JDK: java 1.3.0-C
Output path: Builder4Projects/Amsterdam/classes
Backup path: ts/JBuilder4Projects/Amsterdam/bak
Working directory: C:/WINNT/Profiles/gvdpol/jbproject

IAS 4.1
Interbase

Important for Enterprise JavaBean development is:

- Selecting the required libraries. Make sure the Enterprise JavaBeans library and the IAS4 library (see configuring CORBA) are included.
- Selecting the target JDK™ (JDK 1.3.0.C in this case)

Developing an Enterprise JavaBean in ten steps

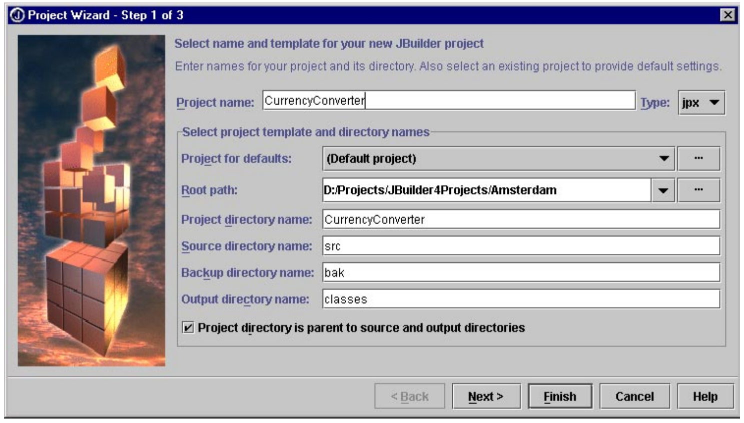
Now that we've set up our JBuilder environment for development of CORBA and EJB components, it's time to start creating our first Enterprise JavaBean.

JBuilder provides several wizards to assist with creating Enterprise JavaBeans, and with generating the required interfaces. This is an example of creating an Enterprise JavaBean. The bean we create will convert a given amount of Dutch guilders (Hfl) into an amount of Euros (Euro).

Step 1. Create a new project

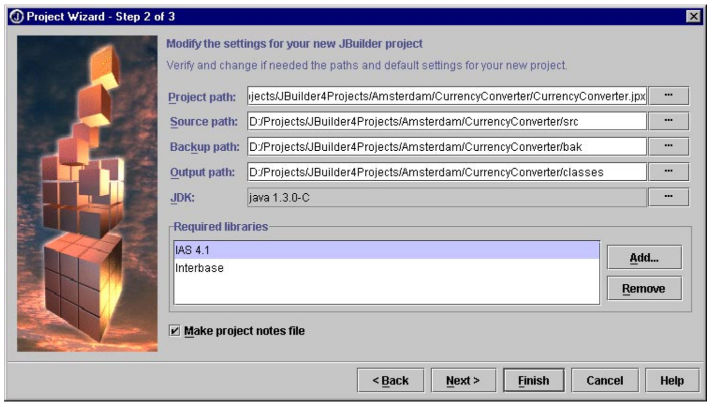
To create an Enterprise JavaBean with JBuilder, first create a new project. To create a new project,

- Select **File|New Project** from the JBuilder menu.
- Enter the project name **CurrencyConverter**.
- Enter the root path of your project.



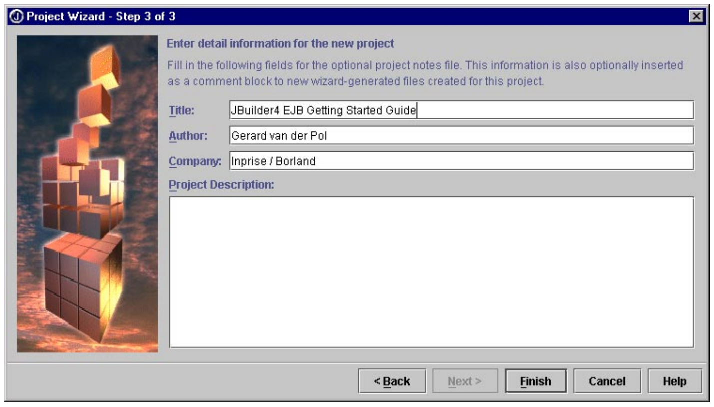
Select name and template for your new JBuilder project
Enter names for your project and its directory. Also select an existing project to provide default settings.
Project name: CurrencyConverter Type: jpx
Select project template and directory names
Project for defaults: (Default project)
Root path: D:/Projects/JBuilder4Projects/Amsterdam
Project directory name: CurrencyConverter
Source directory name: src
Backup directory name: bak
Output directory name: classes
☑ Project directory is parent to source and output directories

The JBuilder 4 Project wizard has been significantly improved; select **<u>N</u>ext** to proceed.



Modify the settings for your new JBuilder project
Verify and change if needed the paths and default settings for your new project.
Project path: jects/JBuilder4Projects/Amsterdam/CurrencyConverter/CurrencyConverter.jpx
Source path: D:/Projects/JBuilder4Projects/Amsterdam/CurrencyConverter/src
Backup path: D:/Projects/JBuilder4Projects/Amsterdam/CurrencyConverter/bak
Output path: D:/Projects/JBuilder4Projects/Amsterdam/CurrencyConverter/classes
JDK: java 1.3.0-C
Required libraries
IAS 4.1
Interbase
☑ Make project notes file

On page two, we'll see that JBuilder has added the default required libraries to our project.We now can add additional libraries if so needed.

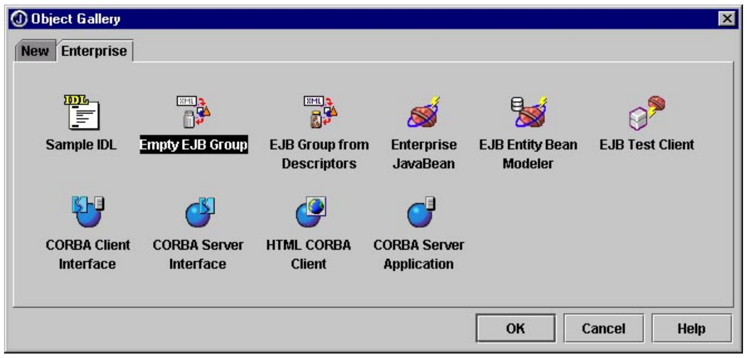

Project Wizard - Step 3 of 3
Enter detail information for the new project
< Back
Next >
Finish
Cancel
Help

Fill in the following fields for the optional project notes file. This information is also optionally inserted as a comment block to new wizard-generated files created for this project.

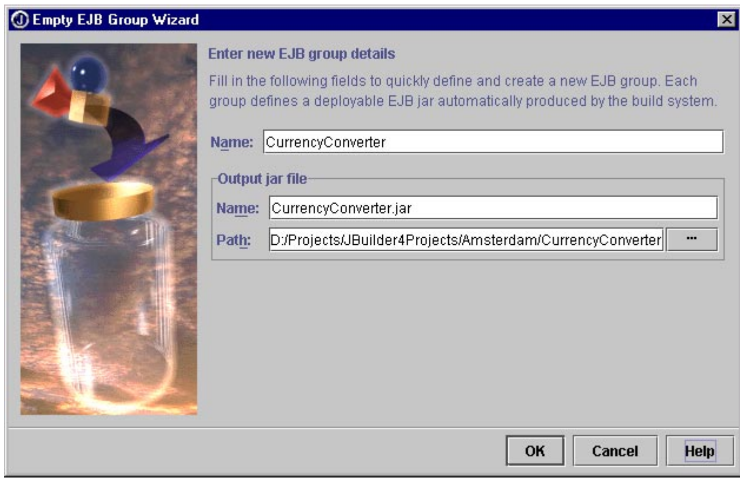
Title: JBuilder4 EJB Getting Started Guide
Author: Gerard van der Pol
Company: Inprise / Borland
Project Description:

On step 3 of the Wizard enter the Title, Author, Company, and Description fields. This is not required. Click **Finish**.

Step 2. Create an Empty EJB Group



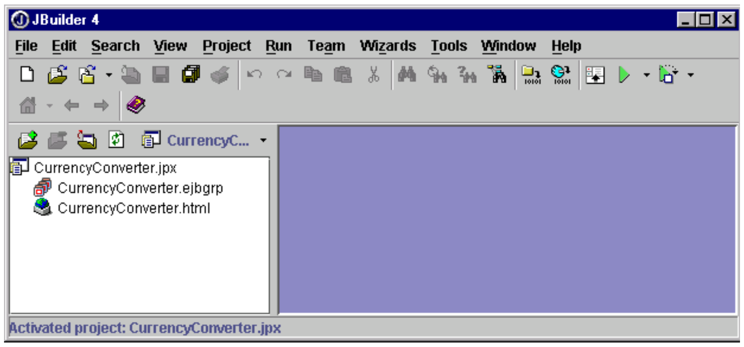
EJB Group is a new feature of JBuilder4. JBuilder Help says that an EJB Group is a logical grouping of one or more beans that will be deployed into a single jarfile. This offers great flexibility in developing your beans; you now will be able to develop multiple jarfiles in one JBuilder Project.



Fill in the following fields to quickly define and create a new EJB group. Each group defines a deployable EJB jar automatically produced by the build system.

Name: CurrencyConverter
Name: CurrencyConverter.jar
Path: D:/Projects/JBuilder4Projects/Amsterdam/CurrencyConverter

Enter the Name of the EJB Group and click OK. By default this name will also be use as the name for the output jar file.


JBuilder 4
File Edit Search View Project Run Team Wizards Tools Window Help
CurrencyC...

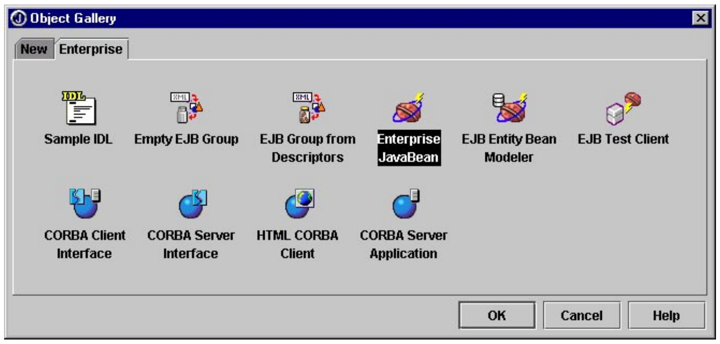
CurrencyConverter.jpx
CurrencyConverter.ejbgrp
CurrencyConverter.html
Activated project: CurrencyConverter.jpx

JBuilder adds an EJB group to our Project.

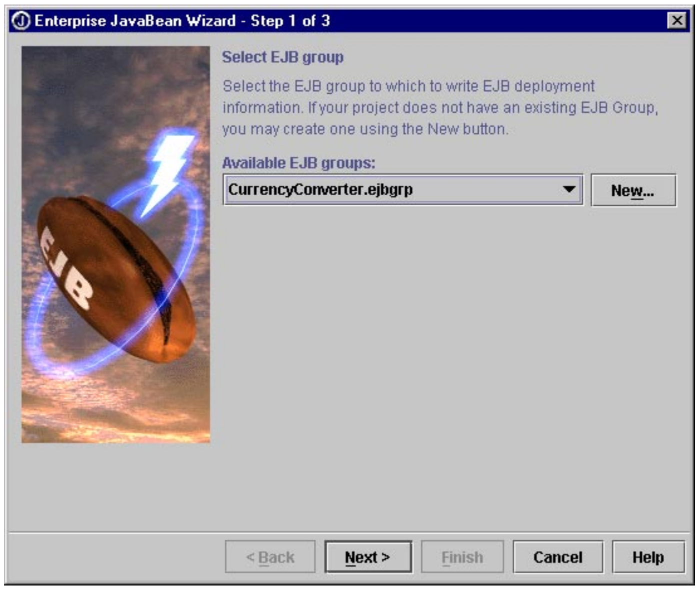
Step 3. Create the Enterprise JavaBean

In this step we are going to create the Enterprise JavaBean.

Select **File|New**, then select Enterprise JavaBean from the Enterprise tab of the Object Gallery.



The Enterprise JavaBean wizard displays.

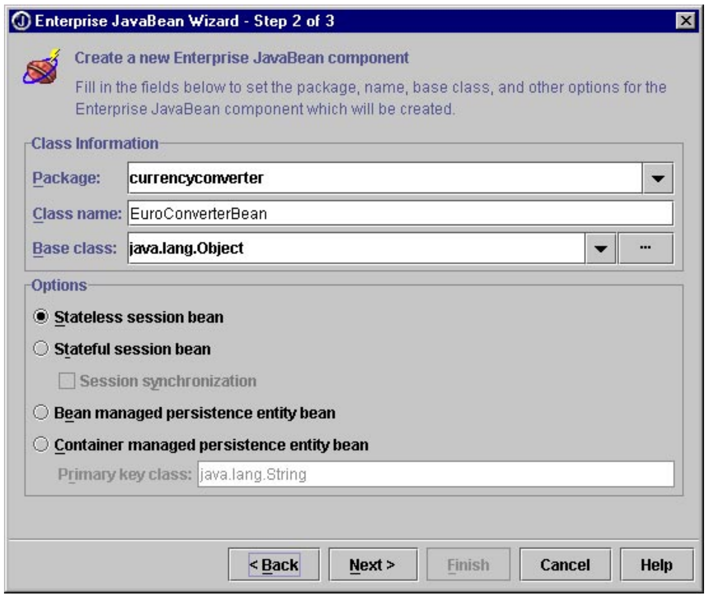

Enterprise JavaBean Wizard - Step 1 of 3

Select EJB group

Select the EJB group to which to write EJB deployment information. If your project does not have an existing EJB Group, you may create one using the New button.

Available EJB groups: CurrencyConverter.ejbgrp

Select the EJB group you've just created and click **Next.** On page two, we can specify the characteristics of our bean.


Enterprise JavaBean Wizard - Step 2 of 3

Create a new Enterprise JavaBean component

Fill in the fields below to set the package, name, base class, and other options for the Enterprise JavaBean component which will be created.

Class Information

Package: currencyconverter

Class name: EuroConverterBean

Base class: java.lang.Object

Options

[✓ Stateless session bean]

Stateful session bean

Session synchronization

Bean managed persistence entity bean

Container managed persistence entity bean

Primary key class: java.lang.String

Do the following:

- Keep the default value for the Package name
- Enter EuroConverterBean™ for the Class name of the new Enterprise JavaBean Component
- Keep the default value of the Base class

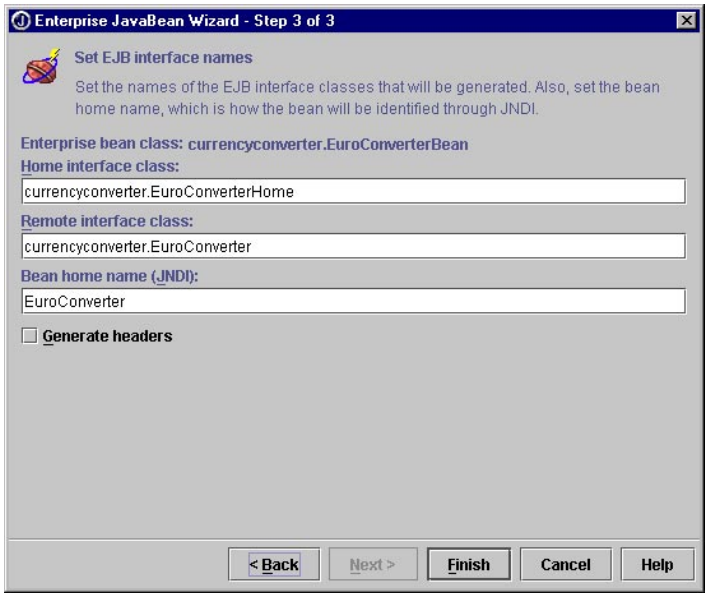
Under the options, select **Stateless session bean** to create a stateless session bean.

EJB 1.1 supports two types of Enterprise JavaBeans:

- Session Beans
  A Session Bean is a bean that is live for one session
- Entity Beans
  An Entity Bean is a bean that maintains its state in, for example, a database

EJB 2.0 adds a new bean, a message bean to the above types. Read the draft specification of EJB 2.0 if you are interested in the new features of this specification; they're available on Sun's website.

Click **Next** to go to the last page of the New EJB Wizard.


Enterprise JavaBean Wizard - Step 3 of 3

Set EJB interface names

Set the names of the EJB interface classes that will be generated. Also, set the bean home name, which is how the bean will be identified through JNDI.

Enterprise bean class: currencyconverter.EuroConverterBean

Home interface class: currencyconverter.EuroConverterHome

Remote interface class: currencyconverter.EuroConverter

Bean home name (JNDI): EuroConverter

Generate headers

This page allows you to enter the names of the Home and Remote Interfaces as well as the JNDI name of our bean. Click **OK** to close the wizard and to create a new bean called EuroConverterBean.java.

JBuilder generates the following source code for Enterprise JavaBean based on the information supplied to the wizard.

```
package currencyconverter;

import java.rmi.*;
import javax.ejb.*;

public class EuroConverterBean implements SessionBean {
  private SessionContext sessionContext;
  public void ejbCreate() {
  }
  public void ejbRemove() {
  }
  public void ejbActivate() {
  }
  public void ejbPassivate() {
  }
  public void setSessionContext(SessionContext context){
    sessionContext = context;
  }
}
```

Step 4. Adding the business logic to the Enterprise JavaBean

Next, we will code the bean by adding a public method signature. Add the following method signature to the EuroConverterBean.java file in the Content pane.

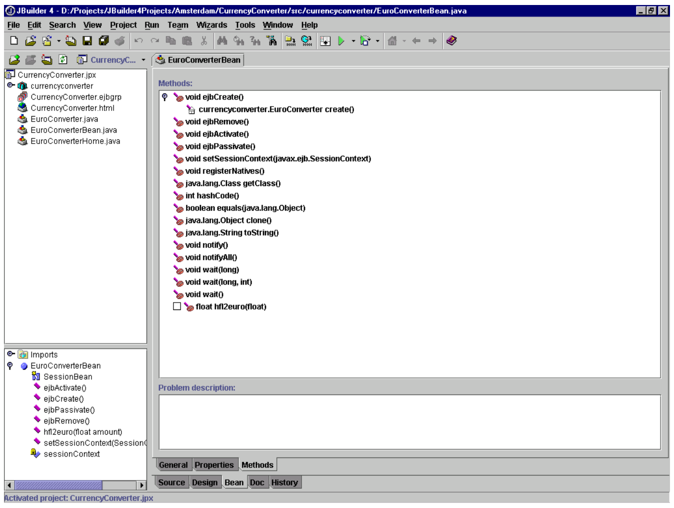
```
public float hfl2euro(float amount) {
    return amount / 2.20371f;
  }
```

Step 5. Refresh the remote interface

Now that the business logic for bean is written, the next step is to make it into an Enterprise JavaBean component by adding the signature of our business method to the remote interface. The home interface defines how the server/container manages the lifecycle of the bean(i.e., creating finding, etc.). The remote interface defines how the client talks to the bean.

To add the signature to the remote interface, select the **Bean** tab of the content tab to activate BeansExpress™.

BeansExpress has been enhanced to support Enterprise JavaBeans as well as regular JavaBeans. On the BeansExpress page, select the methods tab.



JBuilder 4 - D:/Projects/JBuilder4Projects/Amsterdam/CurrencyConverter/src/currencyconverter/EuroConverterBean.java

Methods:

```
void ejbCreate()
    currencyconverter.EuroConverter create()
void ejbRemove()
void ejbActivate()
void ejbPassivate()
void setSessionContext(javax.ejb.SessionContext)
void registerNatives()
java.lang.Class getClass()
int hashCode()
boolean equals(java.lang.Object)
java.lang.Object clone()
java.lang.String toString()
void notify()
void notifyAll()
void wait(long)
void wait(long, int)
void wait()
☐ float hfl2euro(float)
```

Problem description:

Activated project: CurrencyConverter.jpx

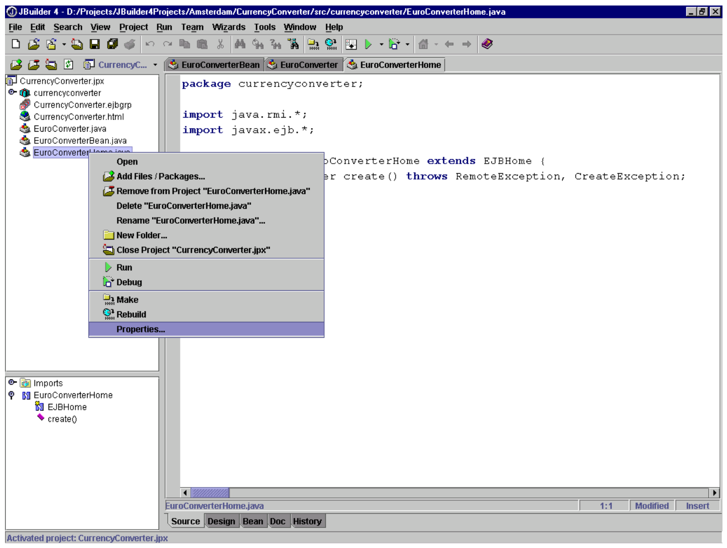
To add our method hfl2euro to the remote interface, check the checkbox in front of the signature. The remote interface, EuroConverterBean.java, now shows our method.

```
package currencyconverter;

import java.rmi.*;
import javax.ejb.*;

public interface EuroConverter extends EJBObject {
  public float hfl2euro(float amount)
      throws RemoteException;
}
```

Step 6. Compiling and generating the stubs and skeletons

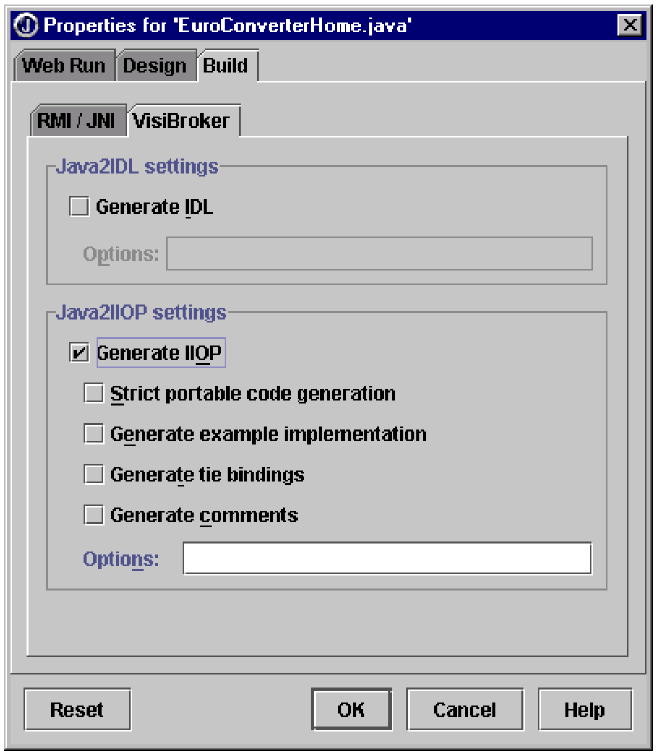
The next step is to compile and generate the IIOP stubs and skeletons. These stubs and skeletons are used to enable the Enterprise JavaBeans to be contacted by RMI over IIOP
(See the Java 2 Enterprise Edition Platform specification on RMI over IIOP)

- ♦ Select the EuroConverterHome.java file.
- ♦ Right-click to produce the popupmenu and choose the "Properties..." menu.



```
package currencyconverter;

import java.rmi.*;
import javax.ejb.*;

                    oConverterHome extends EJBHome {
                    er create() throws RemoteException, CreateException;
```

Select the VisiBroker tab of the Build tab and enable "Generate IIOP" checkbox and Click **OK**

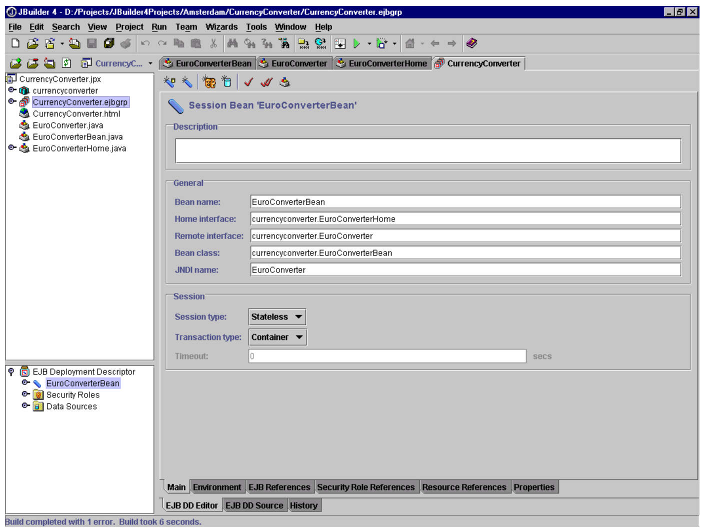


- [ ] Generate IDL
- [✓] Generate IIOP
- [ ] Strict portable code generation
- [ ] Generate example implementation
- [ ] Generate tie bindings
- [ ] Generate comments

Next select **Project|Make Project**. This will start the generation of the IIOP classes based on the Home and Remote interfaces, followed by a compilation of the Java code. During this process, JBuilder 4 generates the jarfile, which we can use to deploy. For this it adds all the necessary files to the jarfile. JBuilder also checks if the deployment descriptor is valid. Because we did not yet complete the deployment descriptor, JBuilder shows an error. We have to specify a transaction attribute for our method, as this is a session bean and the default transaction type is Container. In the next step we will look into the deployment descriptor.

Step 7. Looking at the Deploying Descriptor

By using the Deployment Descriptor Editor™ for editing the deployment descriptor, it's possible to complete the first stage of the deployment. The Deployment Descriptor Editor (DDE) can be used to finish creating the necessary xml deployment description files. The DDE will add the settings, which are defined in the EJB 1.1 specifications, to the ejb-jar.xml file and create an inprise-ejb.xml file containing additional settings needed for deployment, but not defined in the EJB 1.1 specification. Double-click on the ejbgroup to activate the Deployment Descriptor Editor.



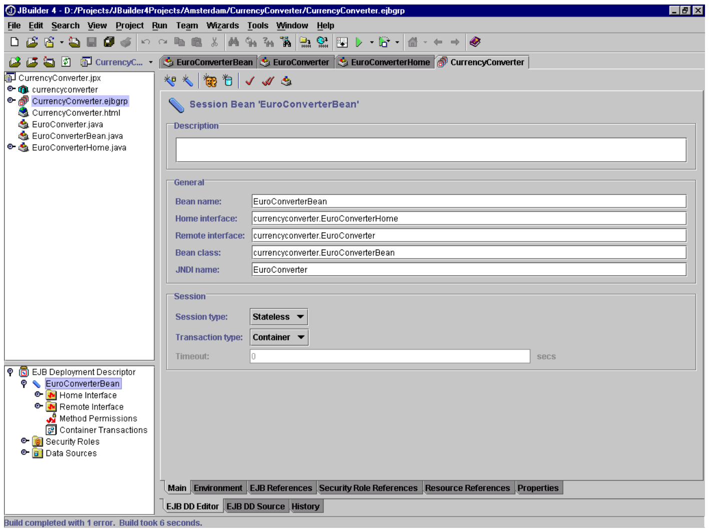
Bean name: EuroConverterBean
Home interface: currencyconverter.EuroConverterHome
Remote interface: currencyconverter.EuroConverter
Bean class: currencyconverter.EuroConverterBean
JNDI name: EuroConverter
Session type: Stateless
Transaction type: Container
Timeout: 0 secs

Build completed with 1 error. Build took 6 seconds.

You now have all the functionality also available in the Inprise Application Server Console. The Editor has the following tabs:

- Main
- Environment
- EJB References
- Security Role References
- Resource References
- Properties

Those of you familiar with JBuilder 3.5 and Inprise Application Server 4.02 will notice that this version offers much more. Also take a look at the console of Inprise Application Server 4.1— this console also offers great flexibility in configuring your environment.



JBuilder 4 - D:/Projects/JBuilder4Projects/Amsterdam/CurrencyConverter/CurrencyConverter.ejbgrp

Session Bean 'EuroConverterBean'

Description

General
Bean name: EuroConverterBean
Home interface: currencyconverter.EuroConverterHome
Remote interface: currencyconverter.EuroConverter
Bean class: currencyconverter.EuroConverterBean
JNDI name: EuroConverter

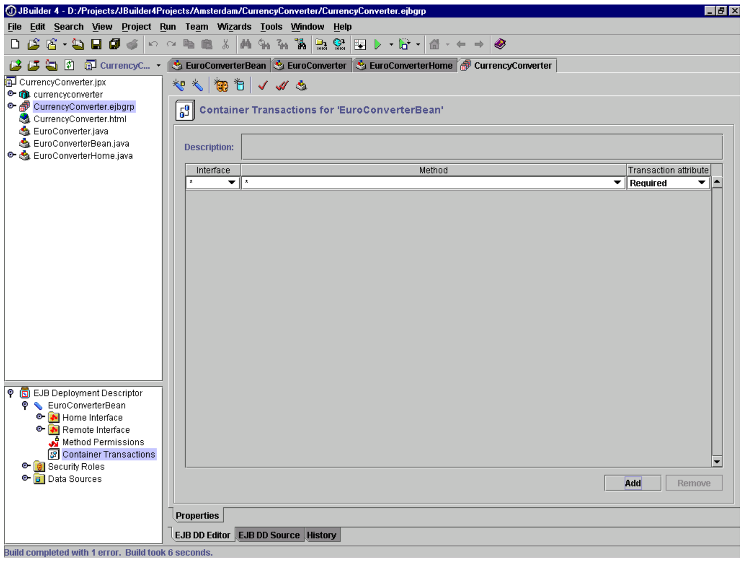
Session
Session type: Stateless
Transaction type: Container
Timeout: 0 secs

Build completed with 1 error. Build took 6 seconds.

First select the bean, and then select the main tab of the editor, as shown in the picture above. The wizard has already entered the JNDI Name for the Enterprise JavaBean. It is also possible to change the session type (stateless or stateful) and the transaction type (Container or Bean managed). We leave the session type to stateless and the transaction type to container.

In order to complete our Enterprise JavaBean, we need to add a container transaction to the assembly descriptor.

Open the nodes of the bean and select the node Container Transactions. Click **Add** to add a new container transaction to our bean deployment descriptor.



JBuilder 4 - D:/Projects/JBuilder4Projects/Amsterdam/CurrencyConverter/CurrencyConverter.ejbgrp

Container Transactions for 'EuroConverterBean'

Description:

Interface	Method	Transaction attribute
*	*	Required

Build completed with 1 error. Build took 6 seconds.

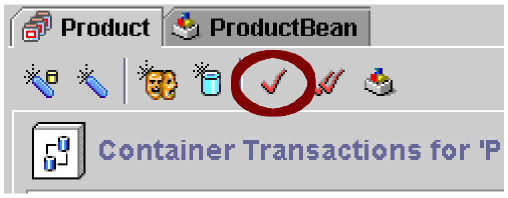
A new container transaction is created.

It is possible to set the transaction attribute. Possible values are:

- Not supported;
- Supports;
- Required;
- RequiresNew;
- Mandatory;
- Never.

For this bean we select Supports, as this is a stateless session bean that is not involved in a transaction. In this case, there is no need to throw an exception if the client requires a transaction. It's possible to make a group of transaction policies for the various interfaces and methods in different combinations.

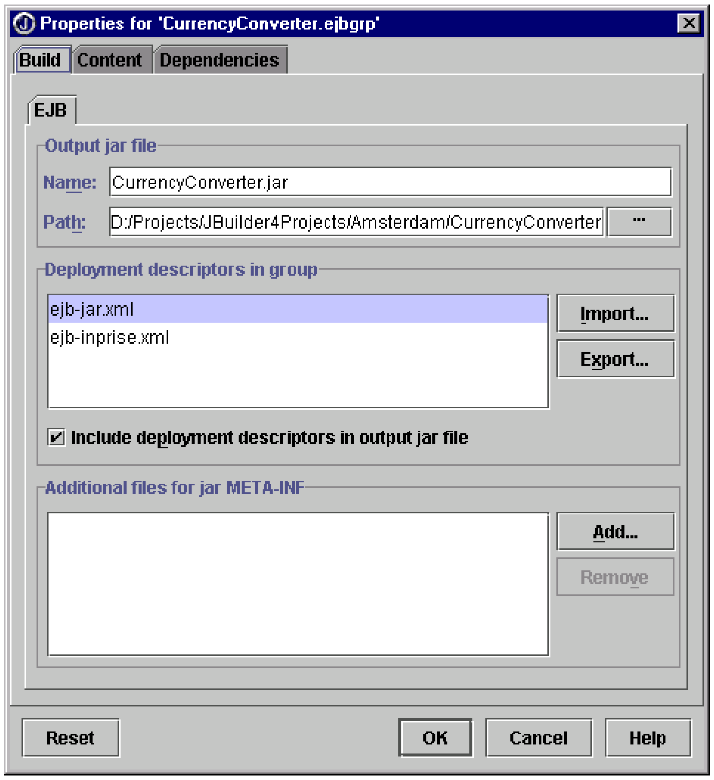
Before we go to the next step, we can verify if the deployment descriptor is ok. Select the root of the structure pane, the item is called EJB deployment descriptor. Right-click this item and select Verify from the popup-menu or click on the button in the top of the content pane as shown below.



The message log window appears and the verification process starts.

Step 8. Create the JAR for the deployment

Next, create a jar containing the Enterprise JavaBean. Because this step is integrated in the build process of JBuilder, we just have to make or rebuild our project.



Properties for 'CurrencyConverter.ejbgrp'

Build | Content | Dependencies

EJB

Output jar file

Name: CurrencyConverter.jar

Path: D:/Projects/JBuilder4Projects/Amsterdam/CurrencyConverter

Deployment descriptors in group

ejb-jar.xml
ejb-inprise.xml

☑ Include deployment descriptors in output jar file

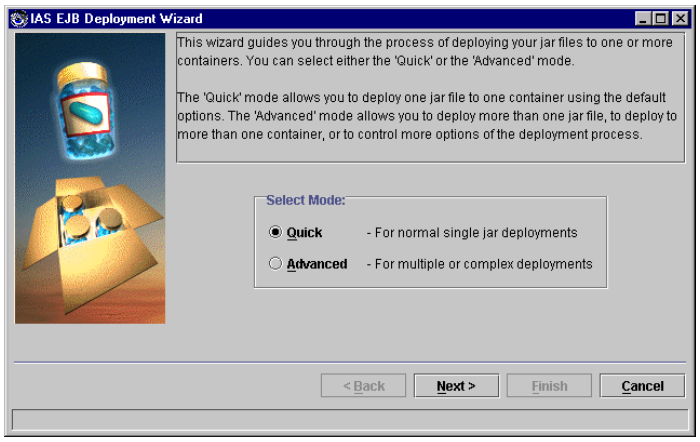
Additional files for jar META-INF

To have complete control over the contents of your JAR file, right-click on the EJB group and select **properties**. The dialog above will appear. You can change the name and location of the jar and have control of the files included. This replaces the dependency wizard in JBuilder 3.5.

Step 9. Deploying the bean to the Inprise Application Server

If you want to deploy your bean, you can use the EJB Deployment tool from the tools menu. This wizard makes it possible to deploy your bean(s) from within JBuilder. In order to successfully deploy your beans, you need to have an application server running.

Select **Tools, EJB Deployment…** to activate it. It's the same wizard as the Deployment wizard, which is activated in the Inprise Application Server Console.



IAS EJB Deployment Wizard

This wizard guides you through the process of deploying your jar files to one or more containers. You can select either the 'Quick' or the 'Advanced' mode.

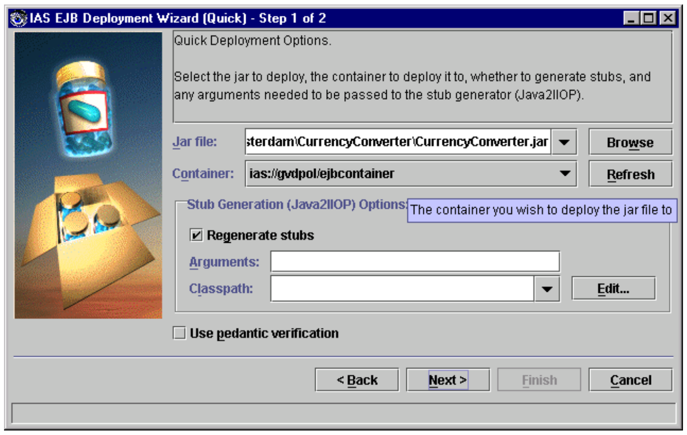
The 'Quick' mode allows you to deploy one jar file to one container using the default options. The 'Advanced' mode allows you to deploy more than one jar file, to deploy to more than one container, or to control more options of the deployment process.

Select Mode:

[✓ Quick] - For normal single jar deployments

Advanced - For multiple or complex deployments

As we have just one bean and one jar file, we will select the quick deployment mode. Click **Next** to go to the next page.



IAS EJB Deployment Wizard (Quick) - Step 1 of 2

Quick Deployment Options.

Select the jar to deploy, the container to deploy it to, whether to generate stubs, and any arguments needed to be passed to the stub generator (Java2IIOP).

Jar file: sterdam\CurrencyConverter\CurrencyConverter.jar

Container: ias://gvdpol/ejbcontainer

The container you wish to deploy the jar file to

Stub Generation (Java2IIOP) Options

☑ Regenerate stubs

Arguments:

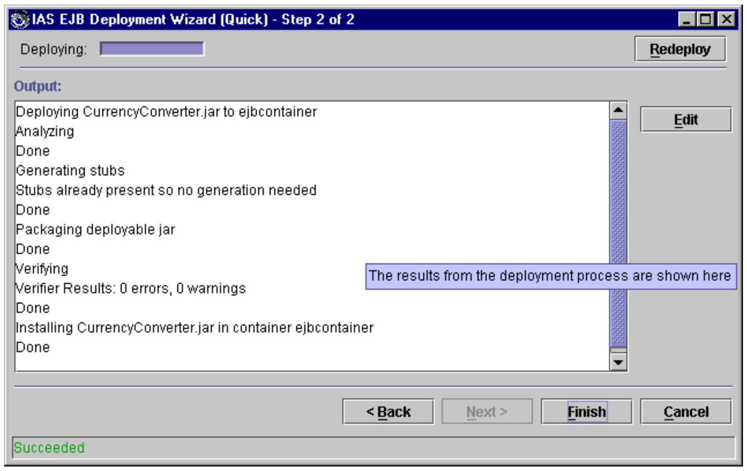
Classpath:

☐ Use pedantic verification

We enter the Quick Deployment wizard. In order to deploy your bean, do the following:

- ♦ Fill in the complete path to your bean or Click **Browse** to navigate to your bean.
- ♦ Select the container to which you want to deploy.
- ♦ Uncheck the Regenerate stubs checkbox. This determines whether the deployment wizard will generate IIOP stub and skeletons. This is already done in JBuilder and packaged in the Jar file.

Click **Next.** This will start the deployment process. The results are shown in the next dialog.



```
Deploying CurrencyConverter.jar to ejbcontainer
Analyzing
Done
Generating stubs
Stubs already present so no generation needed
Done
Packaging deployable jar
Done
Verifying
Verifier Results: 0 errors, 0 warnings
Done
Installing CurrencyConverter.jar in container ejbcontainer
Done
```

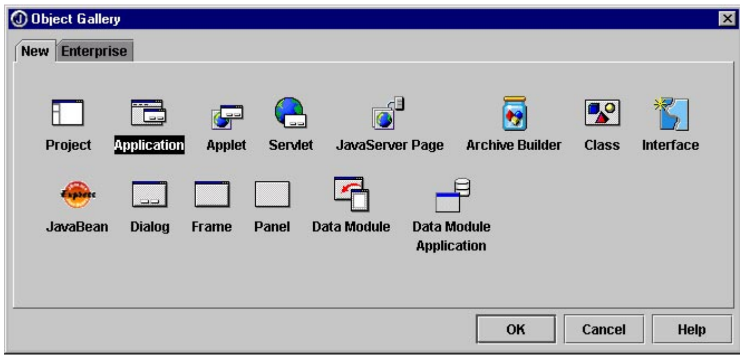
The results from the deployment process are shown here

Succeeded

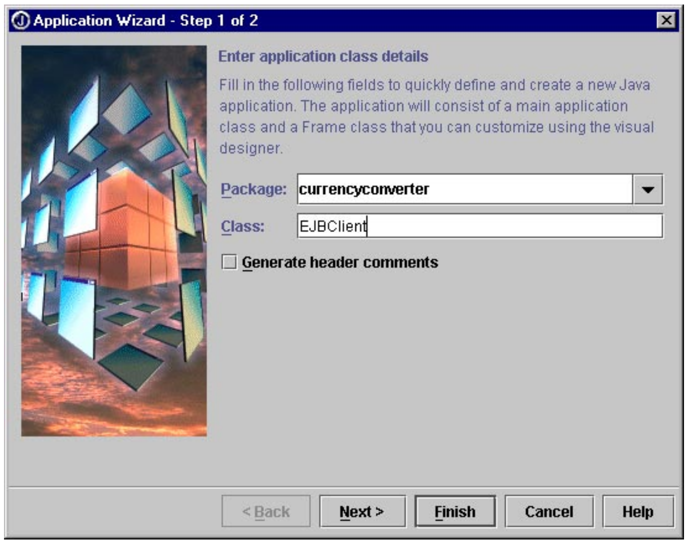
If you want to take an extra look at the deployment descriptor select the **Advanced Deployment Mode**.

Step 10. Creating an Enterprise JavaBean Client Application

In step 10, we are going to create the Client Application. In JBuilder select **File, New** to open the Object Gallery. Select the Application Icon and press **OK**.



The application wizard appears. Change the default Class to EJB Test Client.



Enter application class details

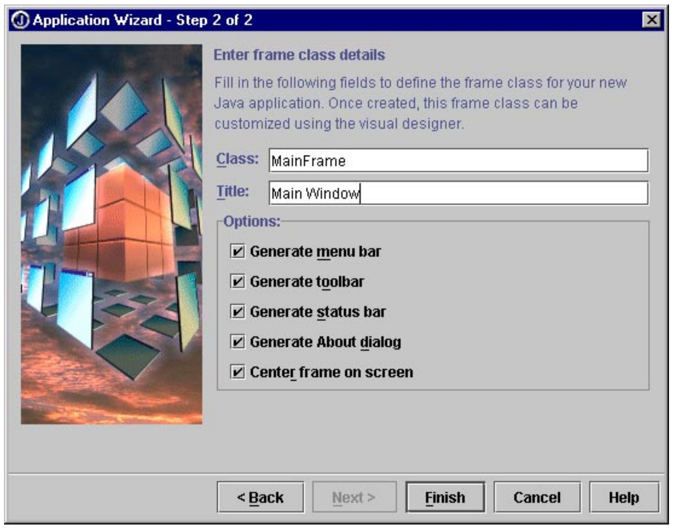
Fill in the following fields to quickly define and create a new Java application. The application will consist of a main application class and a Frame class that you can customize using the visual designer.

Package: currencyconverter

Class: EJBClient

☐ Generate header comments

Click **Next**.



Application Wizard - Step 2 of 2

Enter frame class details

Fill in the following fields to define the frame class for your new Java application. Once created, this frame class can be customized using the visual designer.

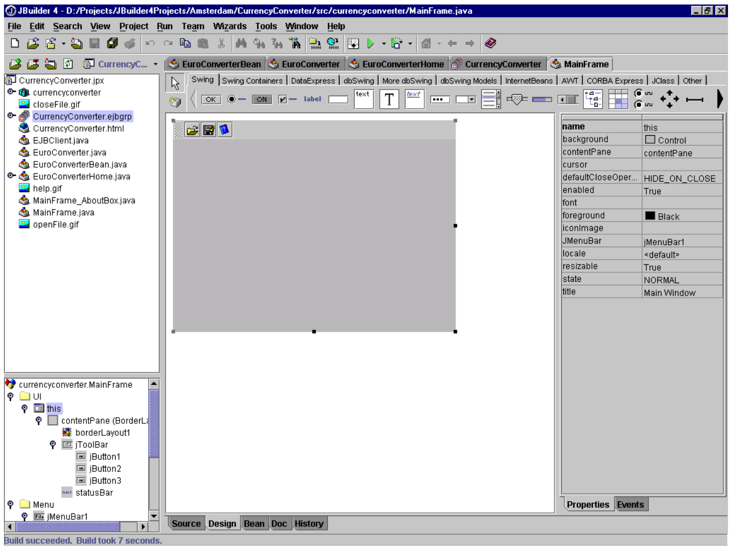
Class: MainFrame

Title: Main Window

Options:

- ☑ Generate menu bar
- ☑ Generate toolbar
- ☑ Generate status bar
- ☑ Generate About dialog
- ☑ Center frame on screen

In step 2 of the wizard, the details for the Frame have to be entered. The name of the class will be **MainFrame** and the title of the frame will be **Main Window**. Select all the options and Click **Finish** to complete this wizard.



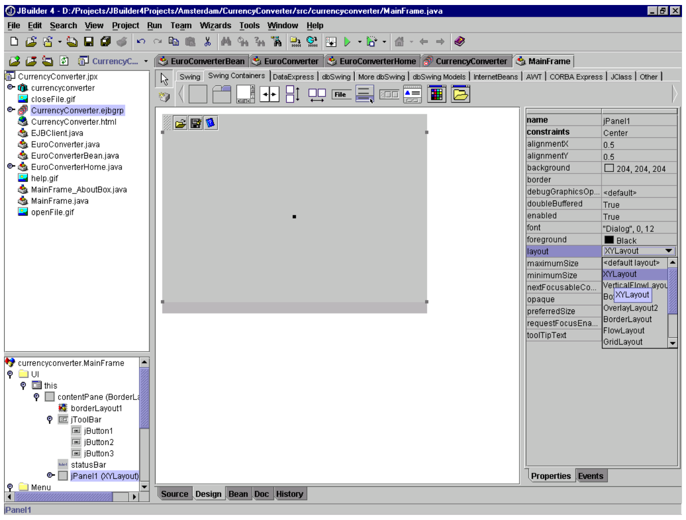
JBuilder 4 - D:/Projects/JBuilder4Projects/Amsterdam/CurrencyConverter/src/currencyconverter/MainFrame.java

name	this
background	⬜ Control
contentPane	contentPane
cursor	
defaultCloseOper...	HIDE_ON_CLOSE
enabled	True
font	
foreground	⬛ Black
iconImage	
JMenuBar	jMenuBar1
locale	<default>
resizable	True
state	NORMAL
title	Main Window

Build succeeded. Build took 7 seconds.

Select the file Mainframe.java in the navigation pane and click on the design. First, we are going to add a panel to the frame. Select the JPanel component from the Swing containers tab of the component palette and drop the component on the frame.

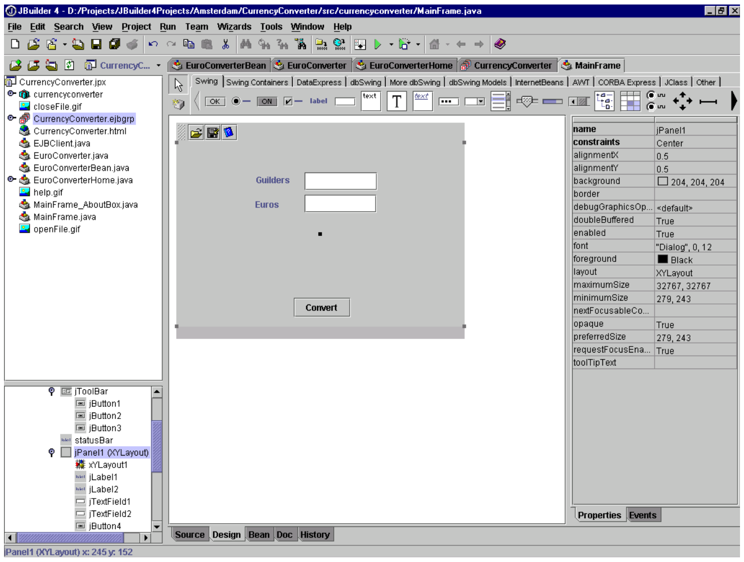
In order to quickly arrange the other components on the frame, we will set the layout of the panel to XYLayout. This gives you more flexibility when developing your frame. When the frame is finished, you can select the layout manager of your choice. To do this, click on the node in the structure pane that represents the panel component and select the layout property in the object inspector. From the combobox select XYLayout.



JBuilder 4 - D:/Projects/JBuilder4Projects/Amsterdam/CurrencyConverter/src/currencyconverter/MainFrame.java

name	jPanel1
constraints	Center
alignmentX	0.5
alignmentY	0.5
background	⬜ 204, 204, 204
border	
debugGraphicsOp...	<default>
doubleBuffered	True
enabled	True
font	"Dialog", 0, 12
foreground	⬛ Black
layout	XYLayout
maximumSize	
minimumSize	
nextFocusableCo...	
opaque	
preferredSize	
requestFocusEna...	
toolTipText	

Next, place two label components, two textfield components, and a button component on the frame. Your frame should look more or less like the picture below.



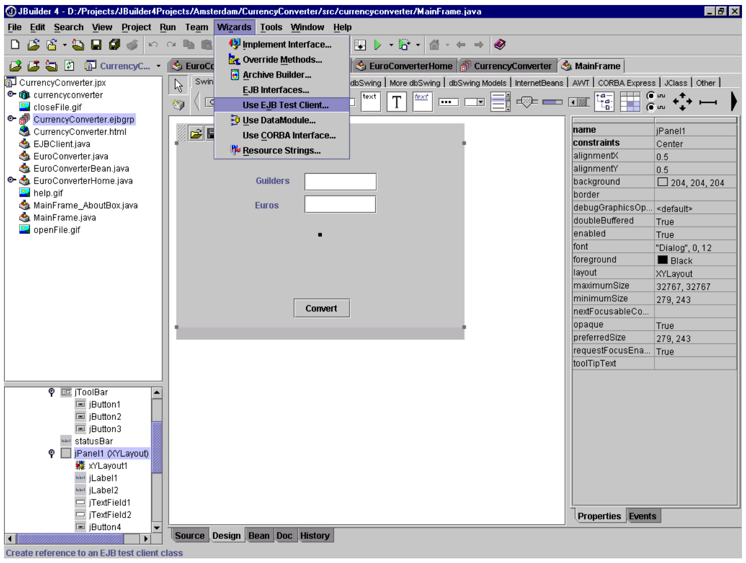
JBuilder 4 - D:/Projects/JBuilder4Projects/Amsterdam/CurrencyConverter/src/currencyconverter/MainFrame.java

name	jPanel1
constraints	Center
alignmentX	0.5
alignmentY	0.5
background	⬜ 204, 204, 204
border	
debugGraphicsOp...	<default>
doubleBuffered	True
enabled	True
font	"Dialog", 0, 12
foreground	⬛ Black
layout	XYLayout
maximumSize	32767, 32767
minimumSize	279, 243
nextFocusableCo...	
opaque	True
preferredSize	279, 243
requestFocusEna...	True
toolTipText	

jPanel1 (XYLayout) x: 245 y: 152

You can choose to use additional panels and layout managers to make your frame more portable, but we will skip this in this session.

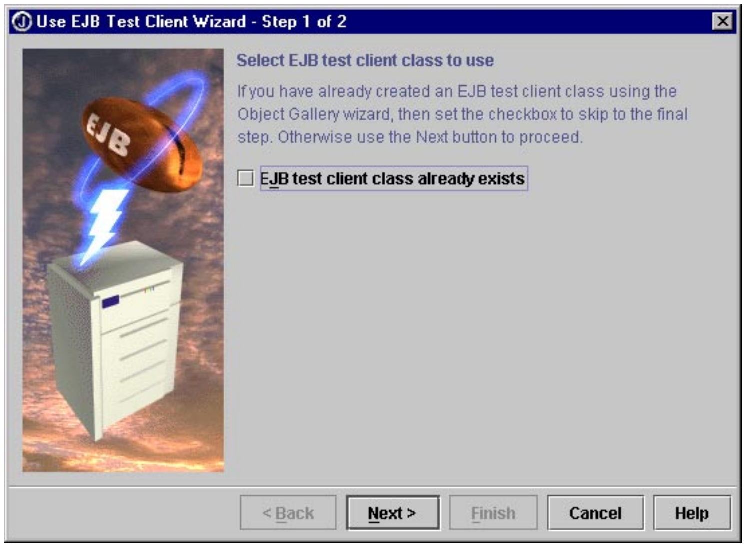
After completing the frame we will add the source for using the Enterprise JavaBeans. Select **wizards, Use EJB Test Client**.


JBuilder 4 - D:/Projects/JBuilder4Projects/Amsterdam/CurrencyConverter/src/currencyconverter/MainFrame.java

name	jPanel1
constraints	Center
alignmentX	0.5
alignmentY	0.5
background	☐ 204, 204, 204
border	
debugGraphicsOp...	<default>
doubleBuffered	True
enabled	True
font	"Dialog", 0, 12
foreground	■ Black
layout	XYLayout
maximumSize	32767, 32767
minimumSize	279, 243
nextFocusableCo...	
opaque	True
preferredSize	279, 243
requestFocusEna...	True
toolTipText	

Create reference to an EJB test client class

The Use EJB Test Client Wizard appears. Because we have not already created an EJB test client class we leave the checkbox open.


Use EJB Test Client Wizard - Step 1 of 2

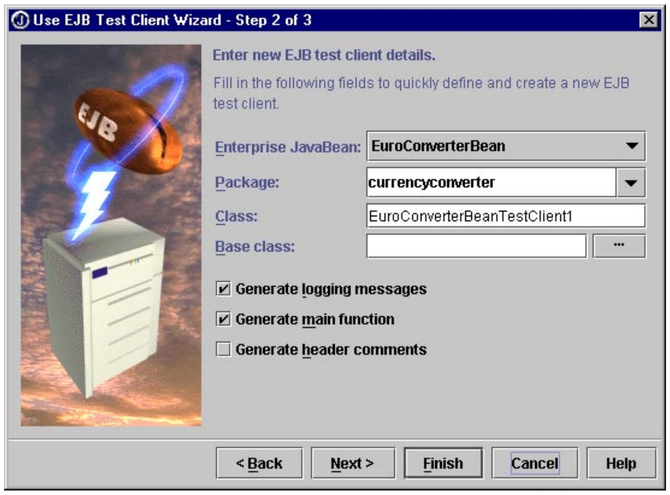
Select EJB test client class to use

If you have already created an EJB test client class using the Object Gallery wizard, then set the checkbox to skip to the final step. Otherwise use the Next button to proceed.

☐ EJB test client class already exists

Click **Next**

On this page we select the EJB we want to use in our frame. As we only have one bean we go for the one JBuilder selected.


Use EJB Test Client Wizard - Step 2 of 3

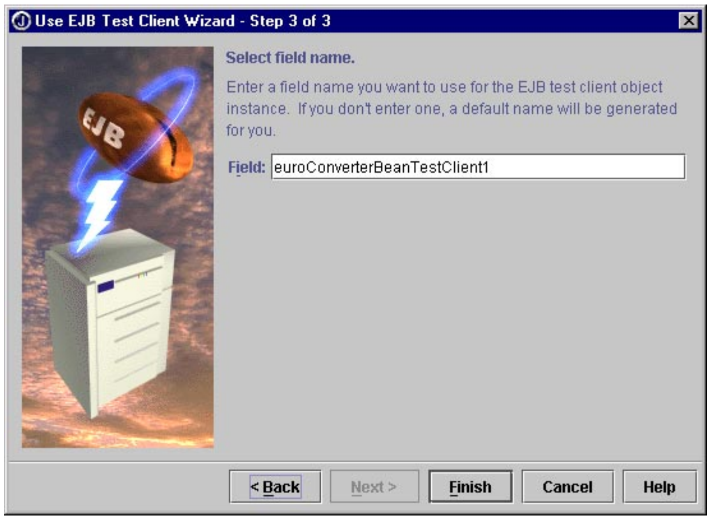
Enter new EJB test client details.

Fill in the following fields to quickly define and create a new EJB test client.

Enterprise JavaBean: EuroConverterBean
Package: currencyconverter
Class: EuroConverterBeanTestClient1
Base class:

☑ Generate logging messages
☑ Generate main function
☐ Generate header comments

Click **Next.**


Use EJB Test Client Wizard - Step 3 of 3

Select field name.

Enter a field name you want to use for the EJB test client object instance. If you don't enter one, a default name will be generated for you.

Field: euroConverterBeanTestClient1

Enter the name of the field JBuilder will insert in your frame; this name belongs to the instance of the EJB test client class. An EJB Test Client class is a JavaBean wrapper, which will act as a proxy for our Enterprise JavaBean. Click Finish to complete the wizard.

JBuilder has created the EJB Test Client class and inserted the field in our frame.

The last thing to do is to activate our bean from the frame. Double-click the Button and JBuilder will create an empty event handler for us.

```
void jButton4_actionPerformed(ActionEvent e) {
    float EuroValue = 0;
    float amount = new
         Float(jTextField1.getText()).floatValue();

    euroConverterBeanTestClient1.create();
    EuroValue =
         euroConverterBeanTestClient1.hfl2euro(amount);

    jTextField2.setText(Float.toString(EuroValue));
  }
```

This completes our client application. As we already deployed our bean, we check if our code is correct. Right-click the EJBClient (our application) and select run from the context menu. The frame should appear and we can enter a value for the guilders we want to convert.

Click on the Convert Button to activate our Stateless Session Bean.

Congratulations!

You have now created, deployed, and invoked your first Enterprise JavaBean. This would be a good moment to expand your background knowledge on Enterprise JavaBean development (and other Java 2 Enterprise Edition™ features) and see how Borland technology can facilitate you in developing your robust and scalable enterprise applications as well as managing them from a distributed application point of view using Inprise AppCenter.

See the web sites mentioned in the Additional information chapter at the beginning of this document for background information.

Borland®

100 Enterprise Way
Scotts Valley, CA 95066-3249
www.borland.com | 831-431-1000 | Fax: 831-431-4113