software AG

webMethods EntireX

EntireX Wrapper for Enterprise JavaBeans

Version 9.6

April 2014

webMethods EntireX

This document applies to webMethods EntireX Version 9.6.

Specifications contained herein are subject to change and these changes will be reported in subsequent release notes or new editions.

Copyright © 1997-2014 Software AG, Darmstadt, Germany and/or Software AG USA, Inc., Reston, VA, USA, and/or its subsidiaries and/or its affiliates and/or their licensors..

The name Software AG and all Software AG product names are either trademarks or registered trademarks of Software AG and/or Software AG USA, Inc. and/or its subsidiaries and/or its affiliates and/or their licensors. Other company and product names mentioned herein may be trademarks of their respective owners.

Detailed information on trademarks and patents owned by Software AG and/or its subsidiaries is located at http://documentation.softwareag.com/legal/.

Use of this software is subject to adherence to Software AG's licensing conditions and terms. These terms are part of the product documentation, located at http://documentation.softwareag.com/legal/ and/or in the root installation directory of the licensed product(s).

This software may include portions of third-party products. For third-party copyright notices and license terms, please refer to "License Texts, Copyright Notices and Disclaimers of Third-Party Products". This document is part of the product documentation, located at http://documentation.softwareag.com/legal/ and/or in the root installation directory of the licensed product(s).

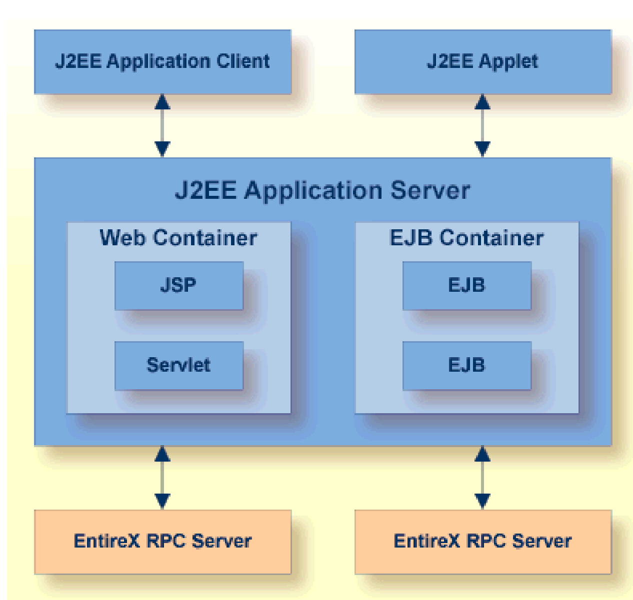
Document ID: EXX-EEXXEJBWRAPPER-96-20140628

Table of Contents

1 Introduction to the EntireX Wrapper for Enterprise JavaBeans 1
2 Using the EntireX Wrapper for Enterprise JavaBeans 3
    Generation Process 4
    Using the Wrapper for EJB Interactively 6
    Generated Classes and Interfaces 23
    Delivered Example 25
3 Using the Wrapper for EJB in Command-line Mode 27
4 Software AG IDL to EJB Mapping 29
    Mapping IDL Data Types to Java Data Types 30
    Mapping Library Name and Alias 31
    Mapping Program Name and Alias 32
    Mapping Parameter Names 32
    Mapping Fixed and Unbounded Arrays 33
    Mapping Groups and Periodic Groups 33
    Mapping Structures 33
    Mapping the Direction Attributes IN, OUT and INOUT 34
5 Writing Applications with the Wrapper for EJB 35
    Programming a Client Application 36
    Compiling and Running the Client Application 37
6 Controlling Applications - EntireX Wrapper for Enterprise JavaBeans 39
    Environment Entries to Control EJB 40
    Using Security/Encryption 41
    Using Natural Security 41
    Using Compression 42
    Using Internationalization with Wrapper for EJB 42
    Tracing 43
    Deployment with an Application Server 43

1 Introduction to the EntireX Wrapper for Enterprise JavaBeans

The EntireX Wrapper for Enterprise JavaBeans enables Java-based components to access an EntireX RPC server using Enterprise JavaBeans. This is accomplished by mapping RPC server libraries to EJB stateful session beans. A remote procedure call will be mapped to a method of the corresponding Enterprise JavaBeans. The Java-based components can use the logic implemented in the libraries of the existing RPC servers, or they can implement new logic on the application server side, using parts of the old logic.



2 Using the EntireX Wrapper for Enterprise JavaBeans

- Generation Process 4
- Using the Wrapper for EJB Interactively 6
- Generated Classes and Interfaces 23
- Delivered Example 25

Generation Process

To generate the Enterprise JavaBeans (EJB) source code, use the *EntireX Workbench*. This can be done interactively with the UI of the *EntireX Workbench*. The generation is controlled by the following properties:

Default Properties for the IDL File

Description	Data Type	Default Value
Broker ID	String	Localhost:1971
Server class	String	RPC
Service class	String	SRV1
Service name	String	CALLNAT
Package name	String	IDL file name without extension
Package name prefix	String	empty string

Generated Files

During the generation process for each library with the name `<libname>`, the following interfaces and classes source files are created in the subdirectory *EJB* of the home directory of the `<name>`*.idl* file.

- The interfaces are created in the subdirectory:

  `<Package Name Prefix><Package Name><file.separator>`*interfaces*.

  They will be a component of the package: `<Package Name Prefix><Package Name>`*.interfaces*

Naming Conventions	Description
EJB`<libname>`*.java*	The remote interface.
EJB`<libname>`*Home.java*	The home interface.

- The EJB classes will be created in the subdirectory:

  `<Package Name Prefix><Package Name><file.separator>`*ejb*

  They will be a component of the package: `<Package Name Prefix><Package Name>`*.ejb*

Naming Conventions	Description
EJB`<libname>`*Bean.java*	The enterprise bean class.
`<libname><innerclassname>`*.java*	Serializable classes for Software AG IDL groups/structs.
`<libname><progname>`*Input.java*	Serializable holder class for all IN and IN/OUT parameters of the program.
`<libname><progname>`*Output.java*	Serializable holder class for all OUT and IN/OUT parameters of the program.
`<libname>`*Mapper.java*	Mapper class.

To build the JAR files for the different application servers, we generate an Ant script which uses the XDoclet tool. This file will be generated in the *EJB* subdirectory:

`<Package Name Prefix><Package Name>`*.xml*

If the package prefix/name contains dots, subdirectories will be created, for example: *abc.def.library* will become *abc/def/library/...*

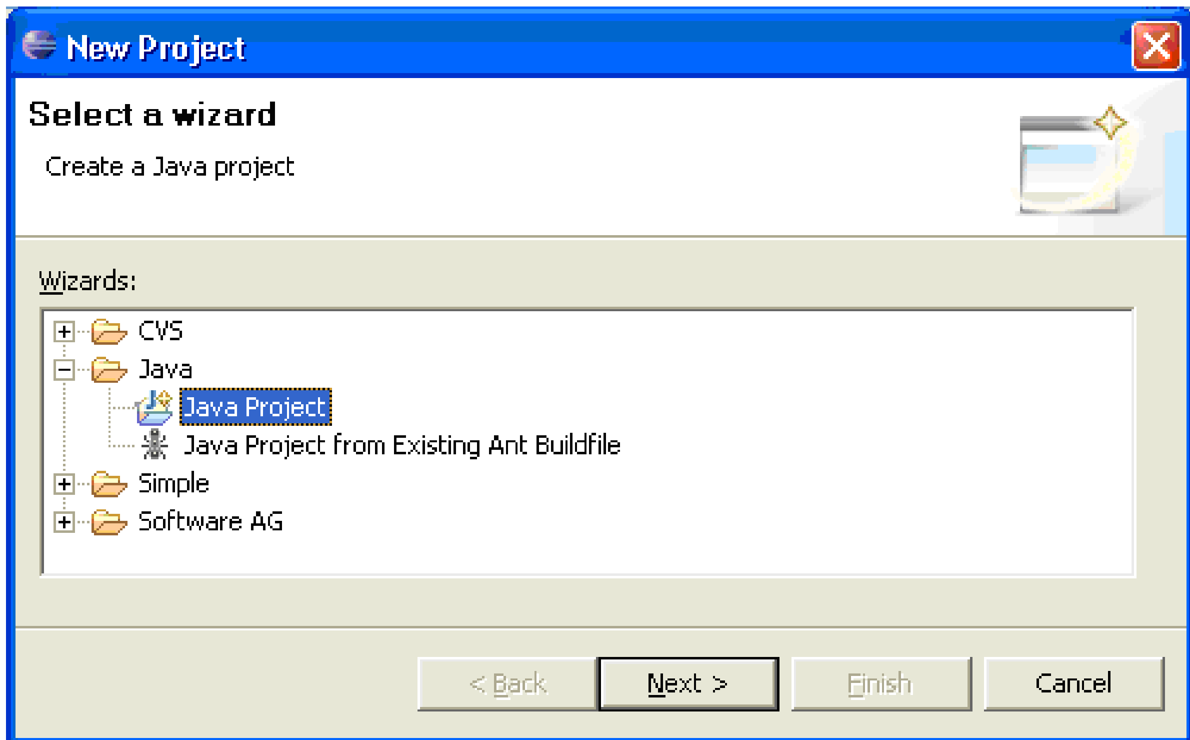
Using the Wrapper for EJB Interactively

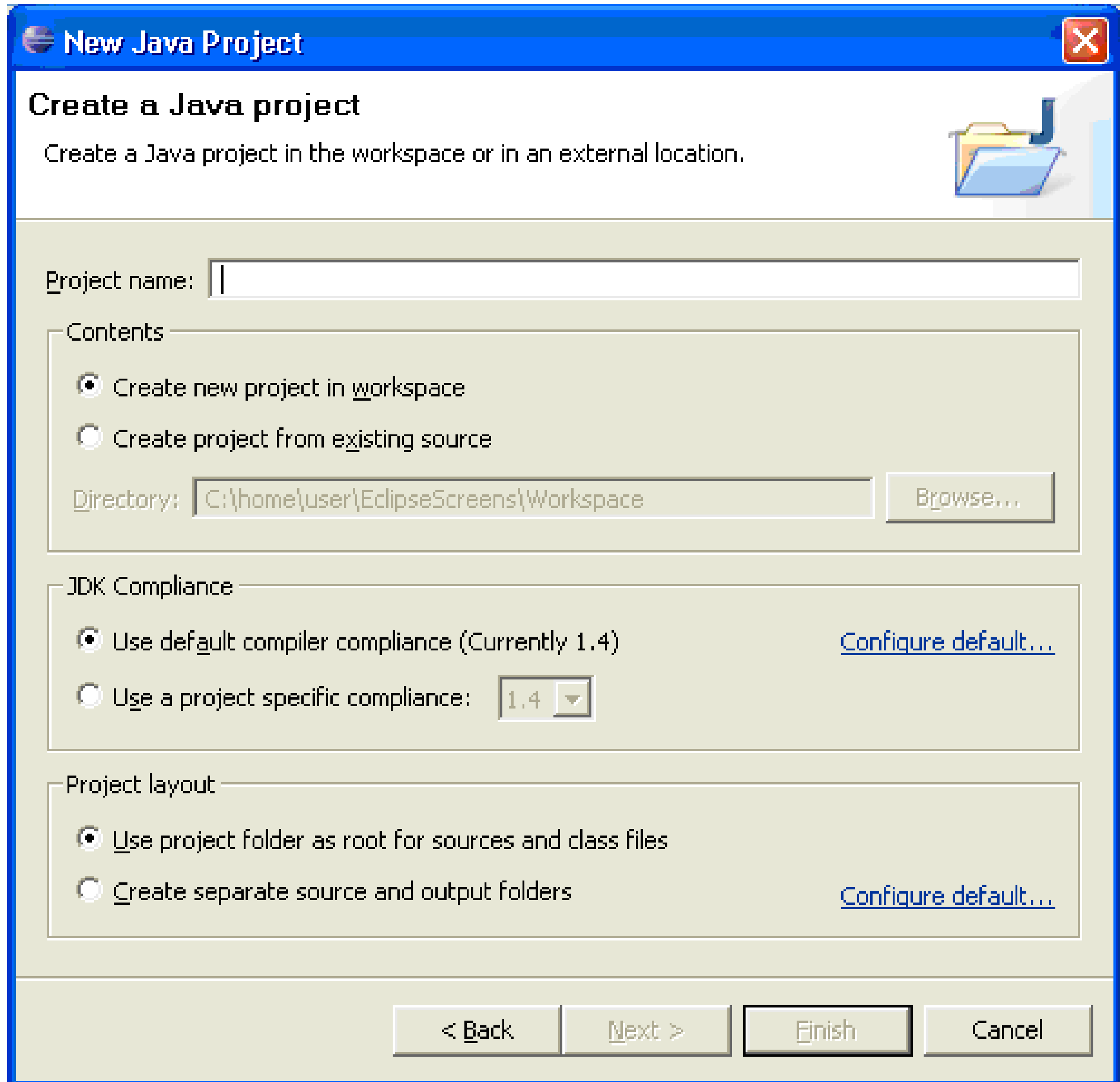
- Using the Wrapper for EJB Functions
- Setting/Modifying EntireX Enterprise JavaBeans Preferences
- Setting/Modifying IDL File Properties

Using the Wrapper for EJB Functions

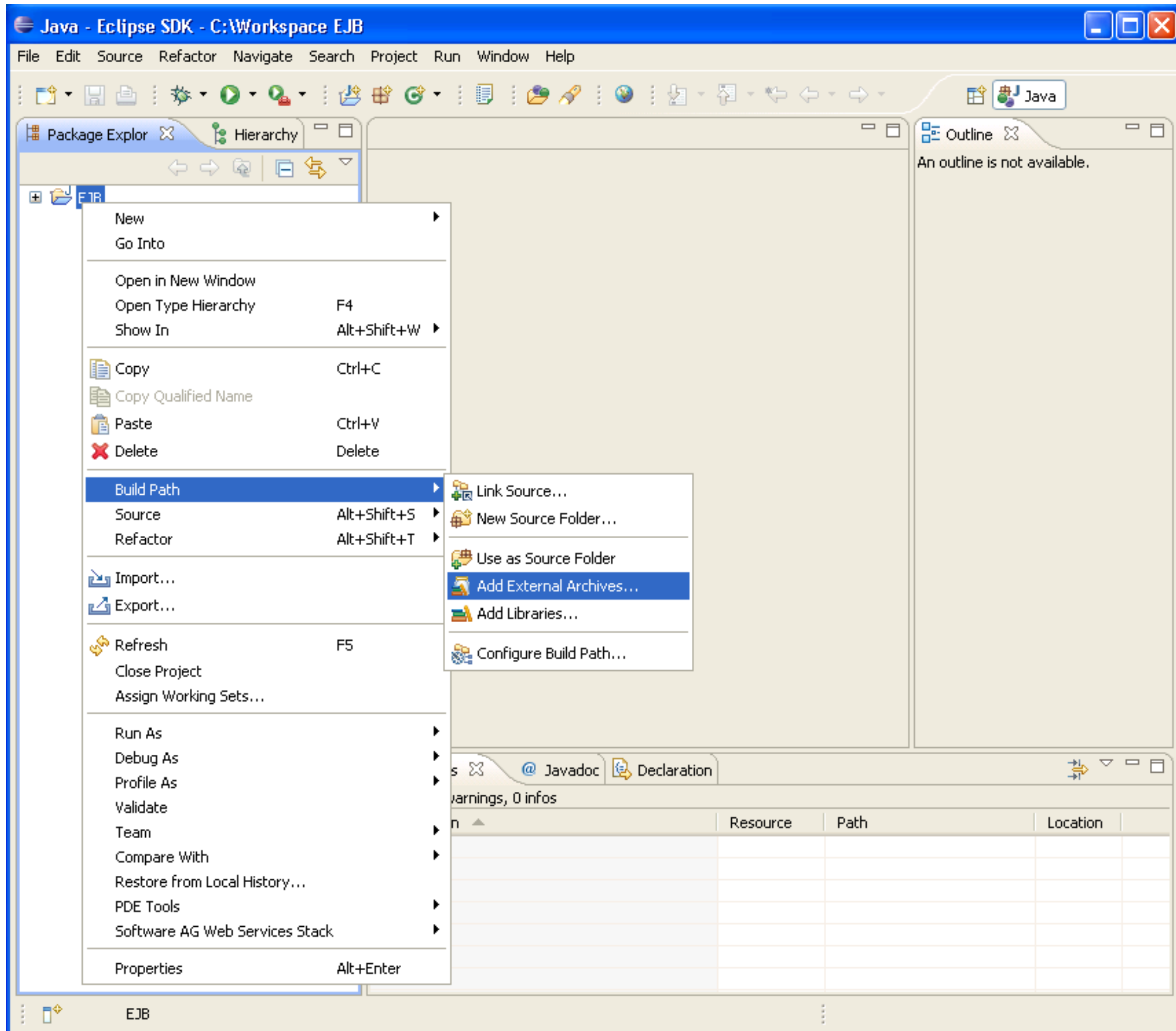
To use the Wrapper for EJB functions

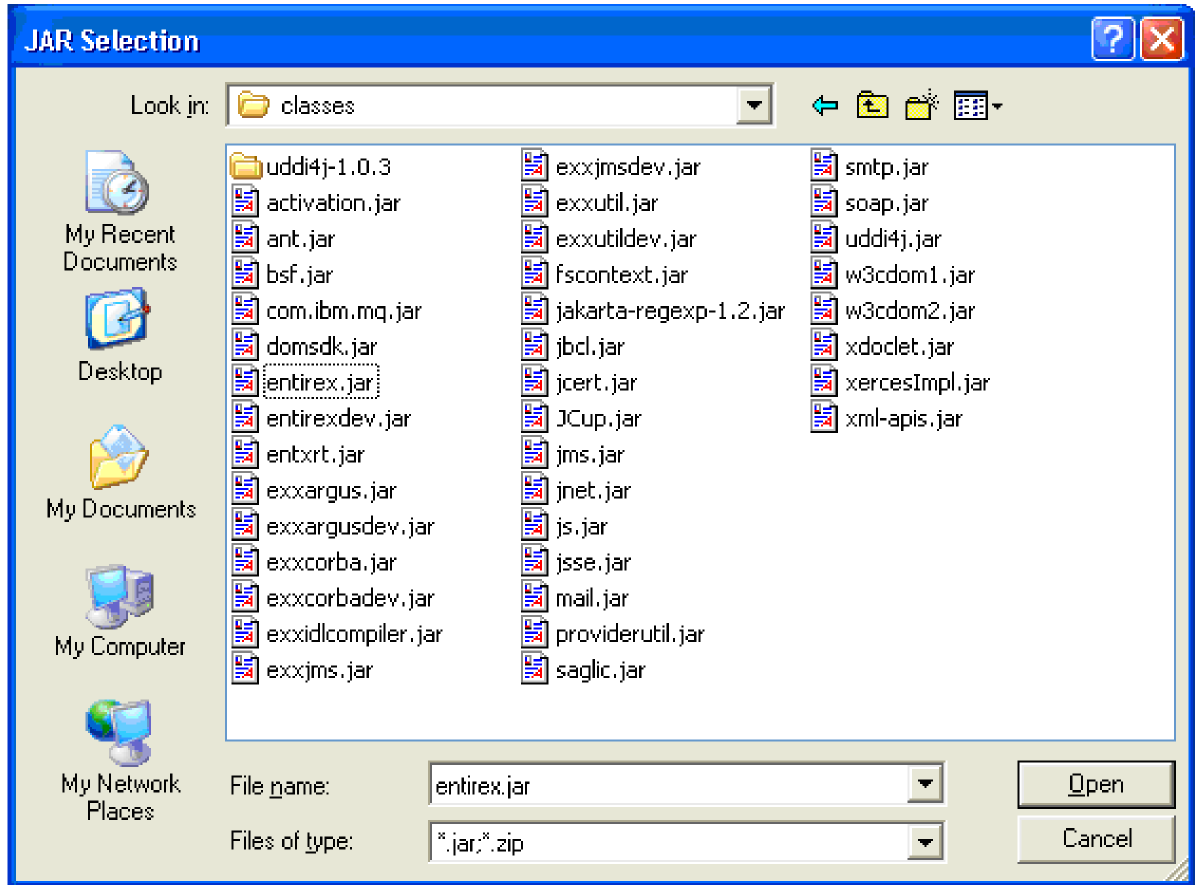
1. Open the *EntireX Workbench*.
2. Create a new Java Project (e.g.: "EJB"), using **File > New > Project**.



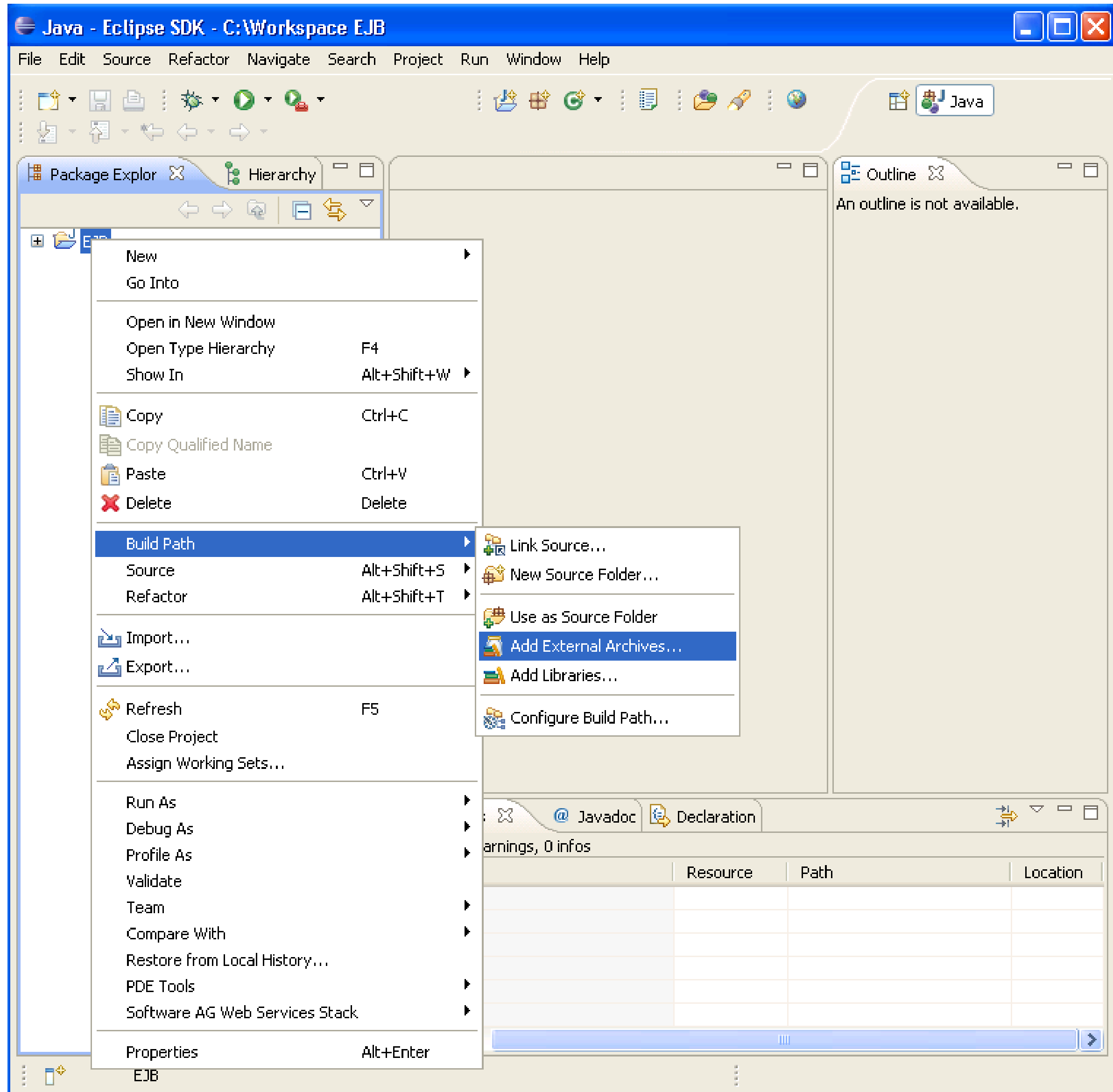


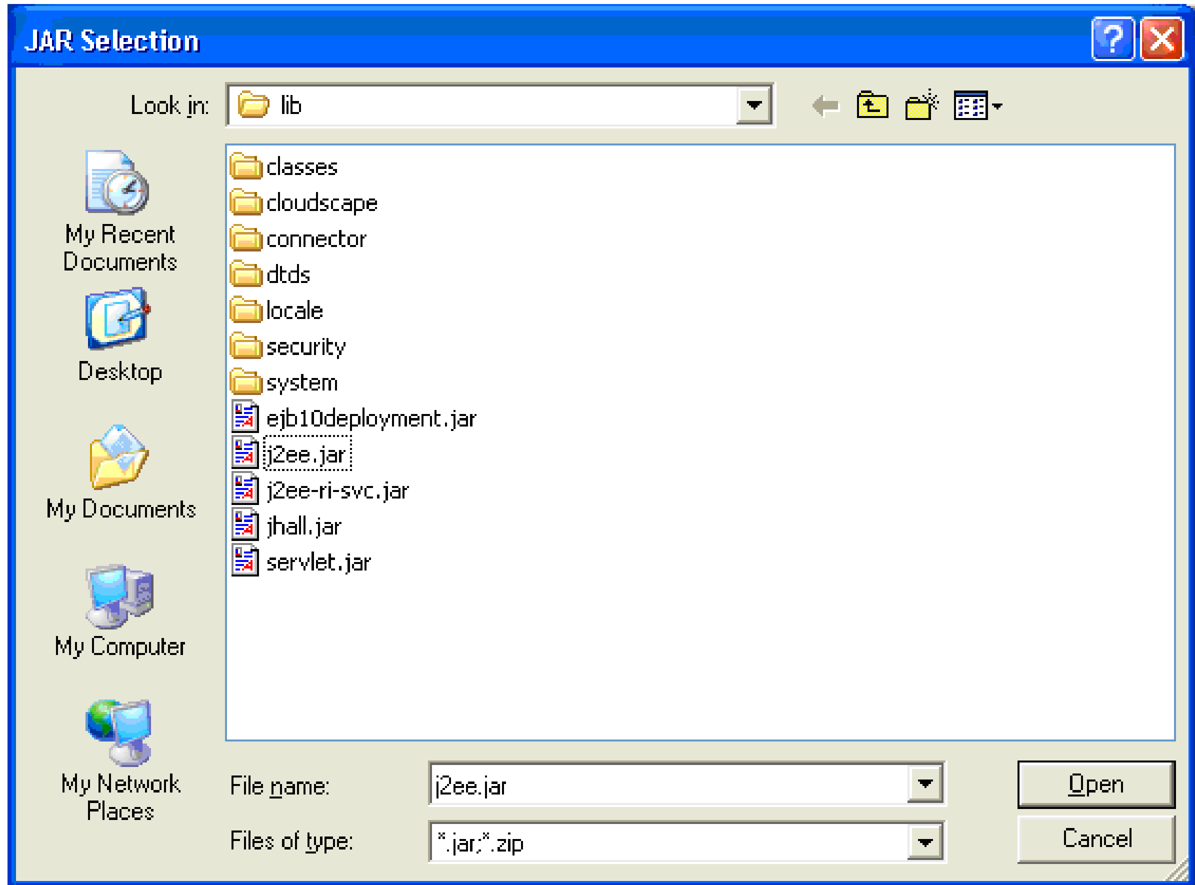
3 Add the EntireX classes to the build path of the project (*entirex.jar*).


Java - Eclipse SDK - C:\Workspace EJB
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explor
Hierarchy
Outline
An outline is not available.
EJB
New
Go Into
Open in New Window
Open Type Hierarchy F4
Show In Alt+Shift+W
Copy Ctrl+C
Copy Qualified Name
Paste Ctrl+V
Delete Delete
Build Path
Source Alt+Shift+S
Refactor Alt+Shift+T
Import...
Export...
Refresh F5
Close Project
Assign Working Sets...
Run As
Debug As
Profile As
Validate
Team
Compare With
Restore from Local History...
PDE Tools
Software AG Web Services Stack
Properties Alt+Enter
Link Source...
New Source Folder...
Use as Source Folder
Add External Archives...
Add Libraries...
Configure Build Path...
Javadoc
Declaration
Resource
Path
Location

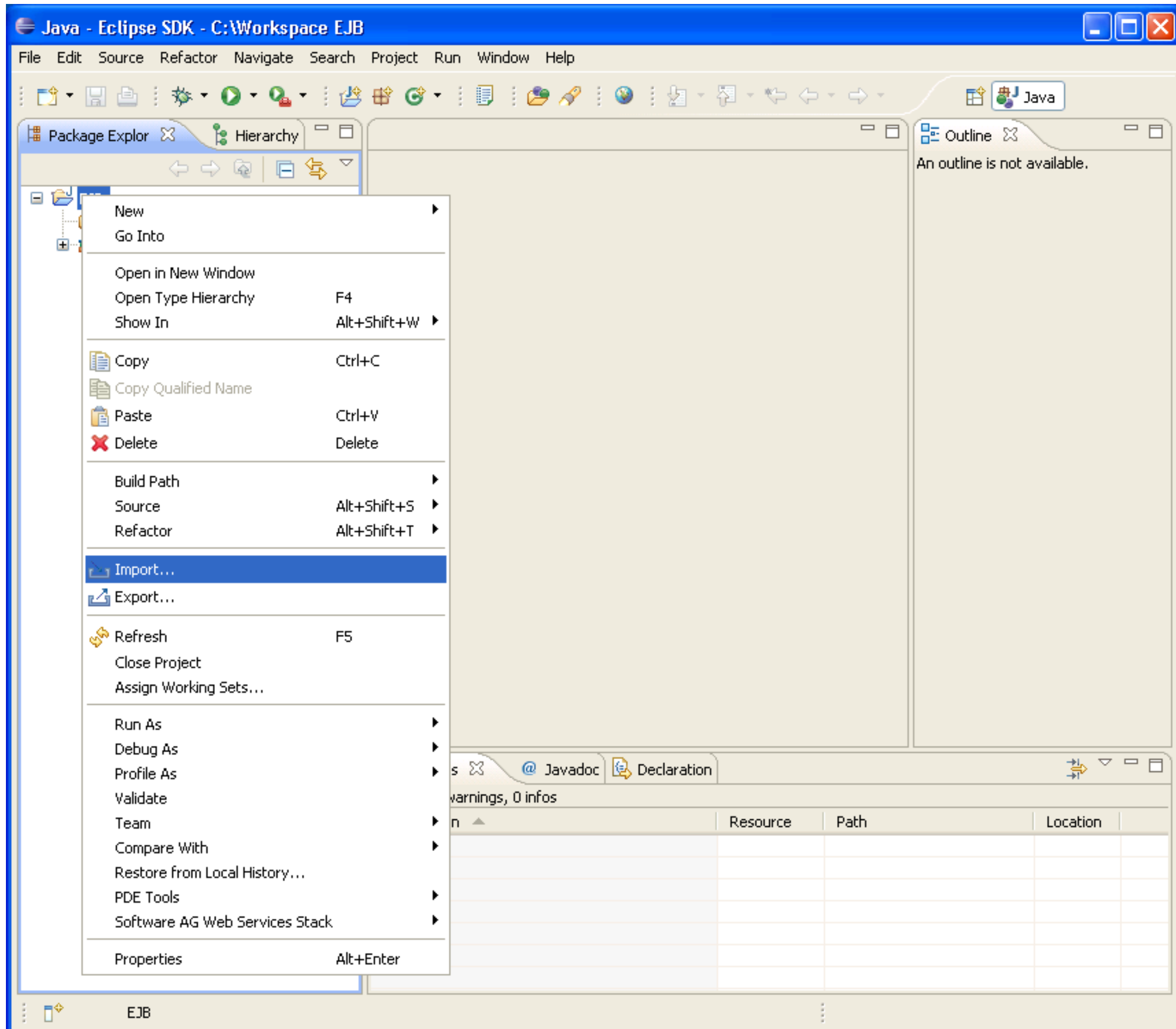


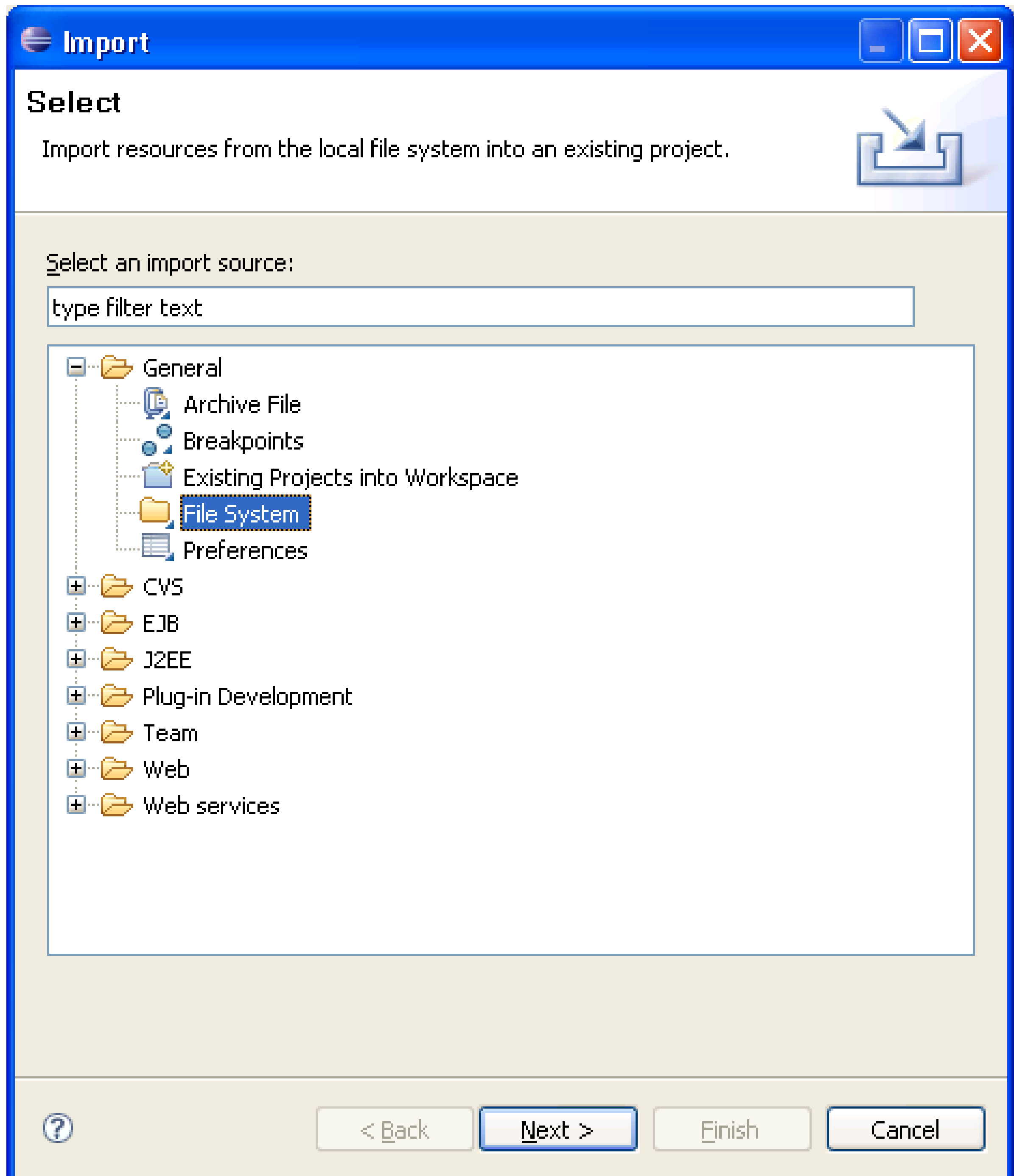
4 Add the Enterprise JavaBeans classes to the build path of the project (e.g.: *j2ee.jar*).

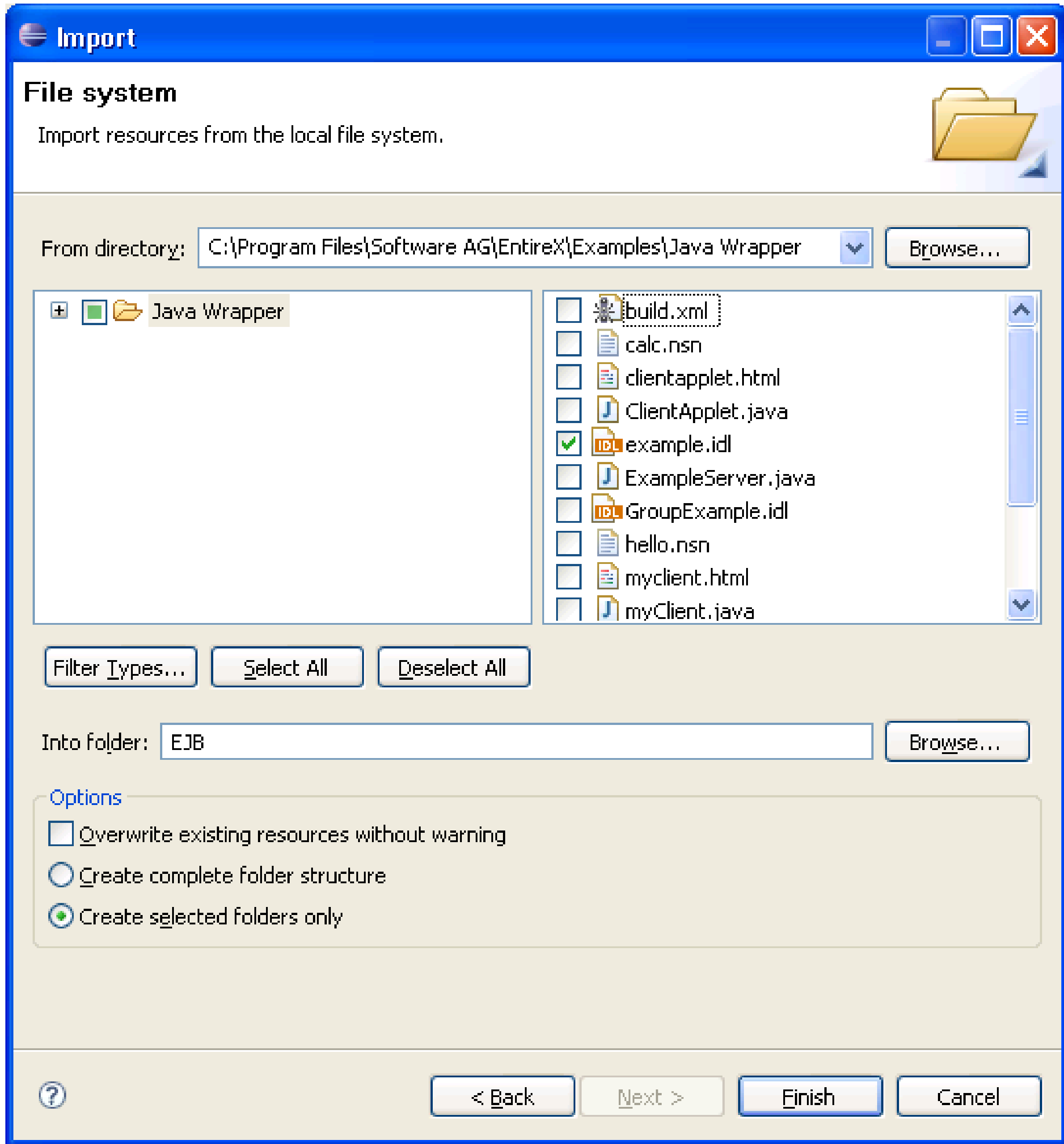

Java - Eclipse SDK - C:\Workspace EJB
File Edit Source Refactor Navigate Search Project Run Window Help
Java
Package Explor
Hierarchy
Outline
An outline is not available.
New
Go Into
Open in New Window
Open Type Hierarchy F4
Show In Alt+Shift+W
Copy Ctrl+C
Copy Qualified Name
Paste Ctrl+V
Delete Delete
Build Path
Source Alt+Shift+S
Refactor Alt+Shift+T
Import...
Export...
Refresh F5
Close Project
Assign Working Sets...
Run As
Debug As
Profile As
Validate
Team
Compare With
Restore from Local History...
PDE Tools
Software AG Web Services Stack
Properties Alt+Enter
Link Source...
New Source Folder...
Use as Source Folder
Add External Archives...
Add Libraries...
Configure Build Path...
Javadoc
Declaration
arnings, 0 infos
Resource
Path
Location
EJB



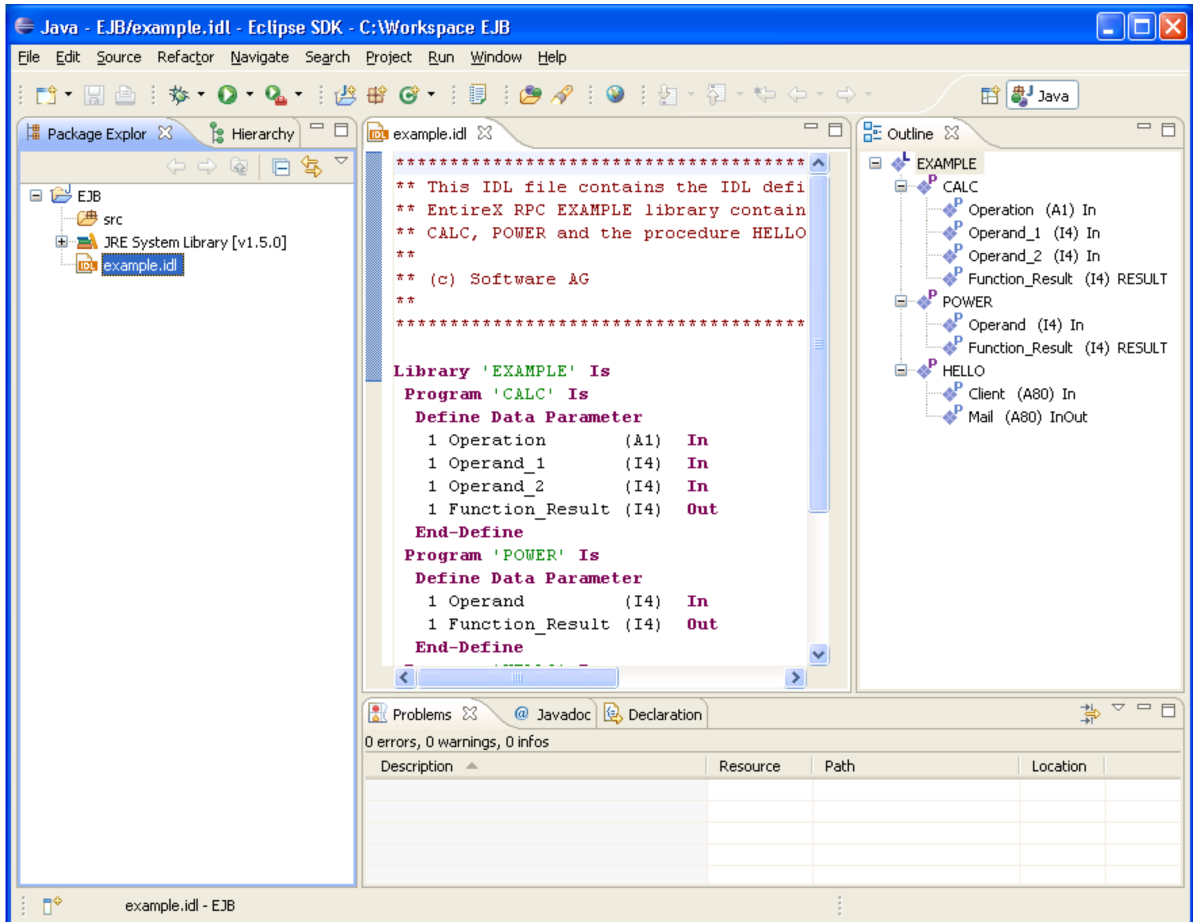
5 Import the IDL file into the project.


Java - Eclipse SDK - C:\Workspace EJB
File Edit Source Refactor Navigate Search Project Run Window Help
Java
Package Explor
Hierarchy
Outline
An outline is not available.
New
Go Into
Open in New Window
Open Type Hierarchy F4
Show In Alt+Shift+W
Copy Ctrl+C
Copy Qualified Name
Paste Ctrl+V
Delete Delete
Build Path
Source Alt+Shift+S
Refactor Alt+Shift+T
Import...
Export...
Refresh F5
Close Project
Assign Working Sets...
Run As
Debug As
Profile As
Validate
Team
Compare With
Restore from Local History...
PDE Tools
Software AG Web Services Stack
Properties Alt+Enter
Javadoc
Declaration
warnings, 0 infos
Resource
Path
Location
EJB


Import
Select
Import resources from the local file system into an existing project.
Select an import source:
type filter text
General
Archive File
Breakpoints
Existing Projects into Workspace
File System
Preferences
CVS
EJB
J2EE
Plug-in Development
Team
Web
Web services
< Back
Next >
Finish
Cancel



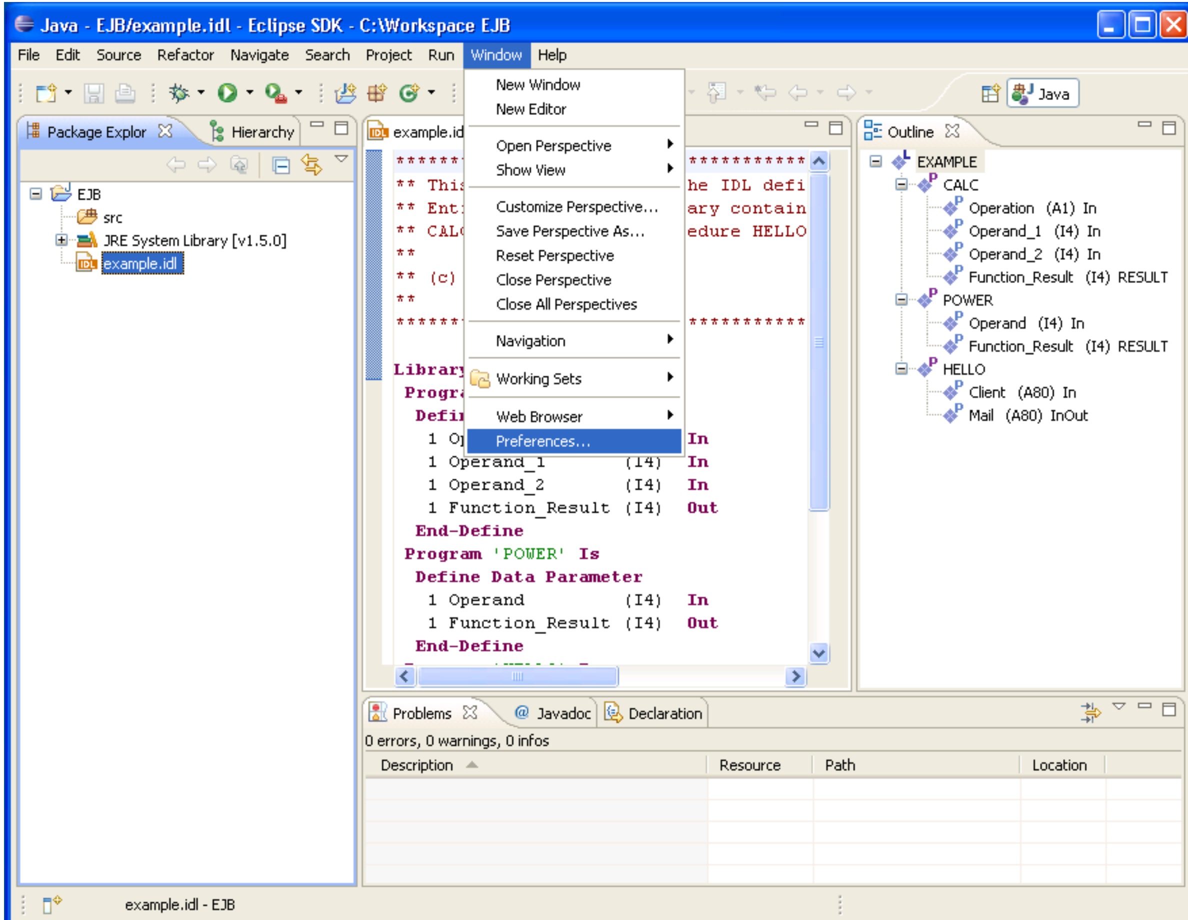
6 Activate the *example.idl* file.

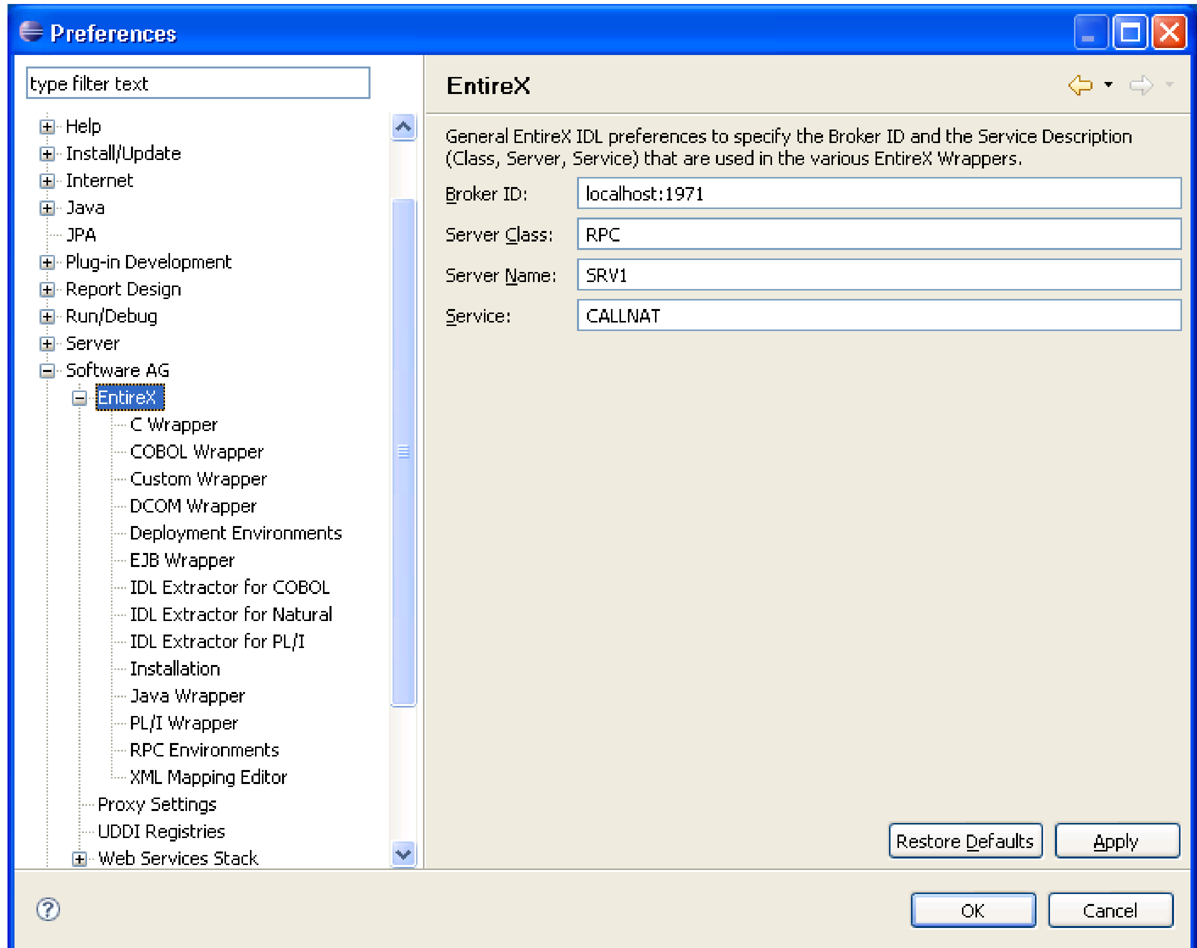


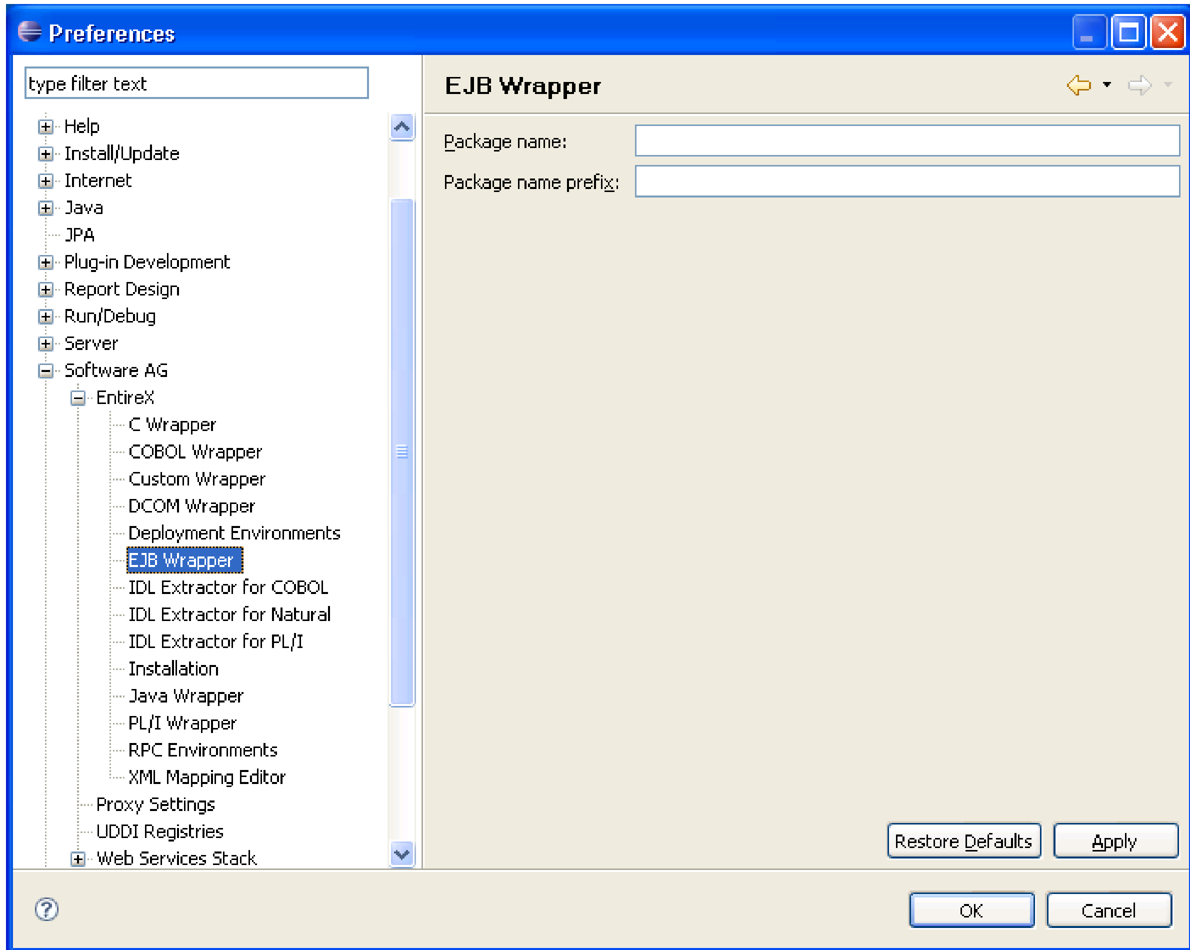
Setting/Modifying EntireX Enterprise JavaBeans Preferences

To set/modify the preferences

- Display the Preferences window.


Java - EJB/example.idl - Eclipse SDK - C:\Workspace EJB
File Edit Source Refactor Navigate Search Project Run Window Help
New Window
New Editor
Open Perspective
Show View
Customize Perspective...
Save Perspective As...
Reset Perspective
Close Perspective
Close All Perspectives
Navigation
Working Sets
Web Browser
Preferences...
Package Explor
Hierarchy
EJB
src
JRE System Library [v1.5.0]
example.idl
Outline
EXAMPLE
CALC
Operation (A1) In
Operand_1 (I4) In
Operand_2 (I4) In
Function_Result (I4) RESULT
POWER
Operand (I4) In
Function_Result (I4) RESULT
HELLO
Client (A80) In
Mail (A80) InOut
Problems
Javadoc
Declaration
0 errors, 0 warnings, 0 infos
Description
Resource
Path
Location
example.idl - EJB

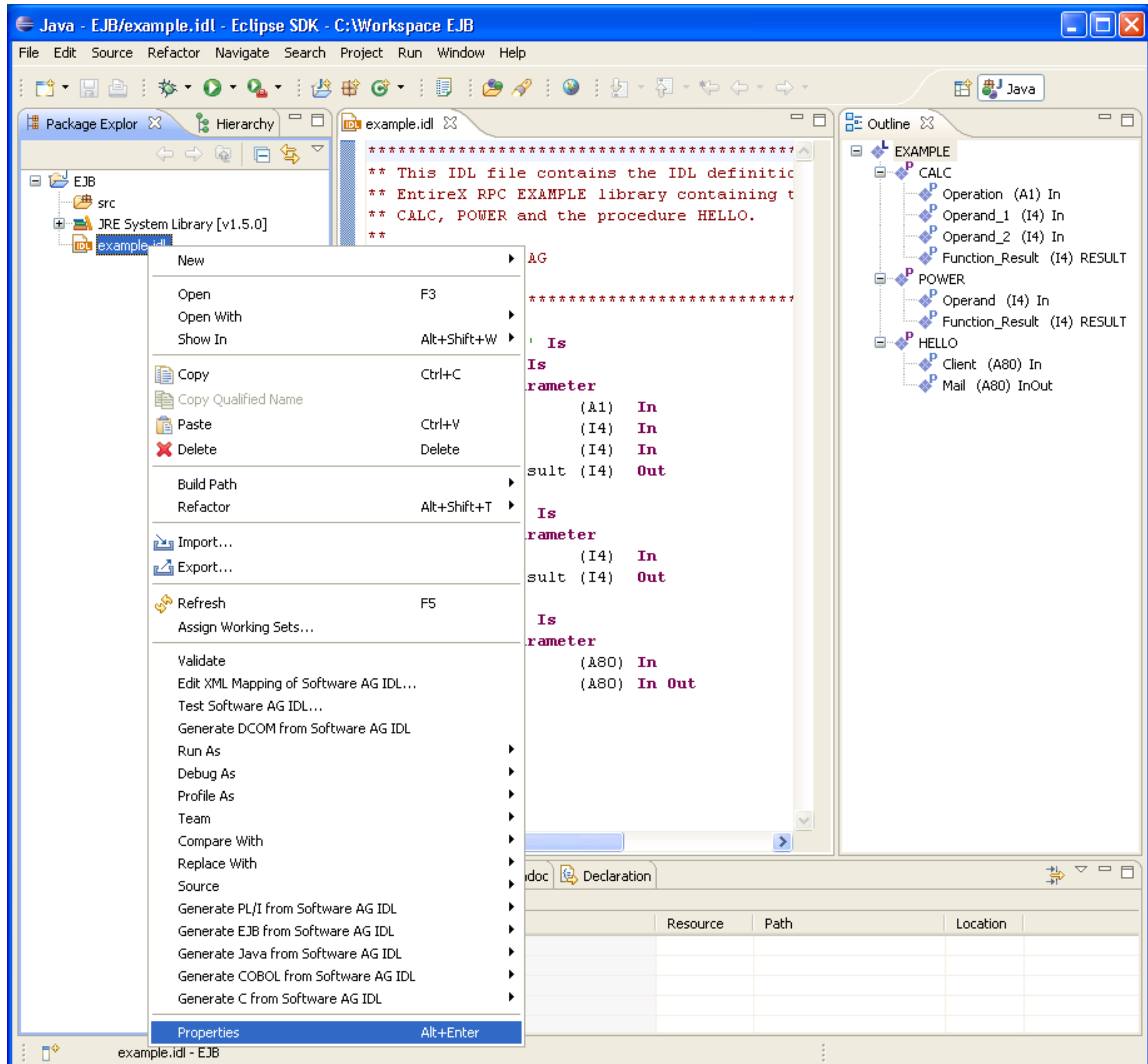

Preferences
type filter text
Help
Install/Update
Internet
Java
JPA
Plug-in Development
Report Design
Run/Debug
Server
Software AG
EntireX
C Wrapper
COBOL Wrapper
Custom Wrapper
DCOM Wrapper
Deployment Environments
EJB Wrapper
IDL Extractor for COBOL
IDL Extractor for Natural
IDL Extractor for PL/I
Installation
Java Wrapper
PL/I Wrapper
RPC Environments
XML Mapping Editor
Proxy Settings
UDDI Registries
Web Services Stack
EntireX
General EntireX IDL preferences to specify the Broker ID and the Service Description (Class, Server, Service) that are used in the various EntireX Wrappers.
Broker ID: localhost:1971
Server Class: RPC
Server Name: SRV1
Service: CALLNAT
Restore Defaults
Apply
OK
Cancel



Setting/Modifying IDL File Properties

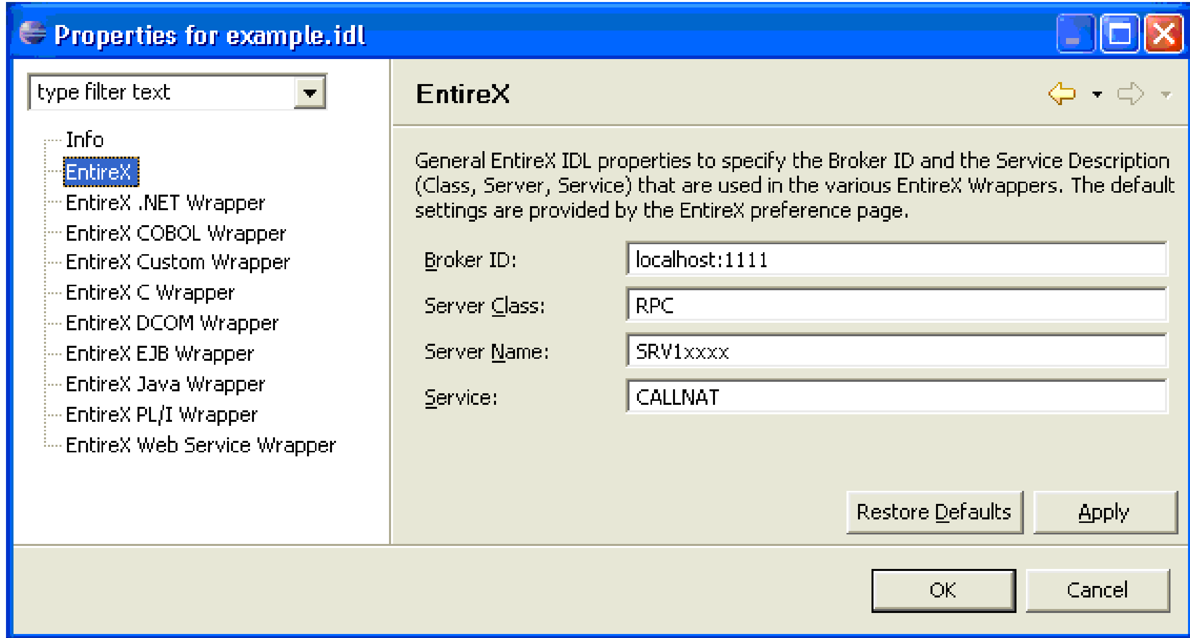
To set/modify the properties

1. Display the Properties window; use either the pop-up menu or, from the **File** menu, choose **Properties**.



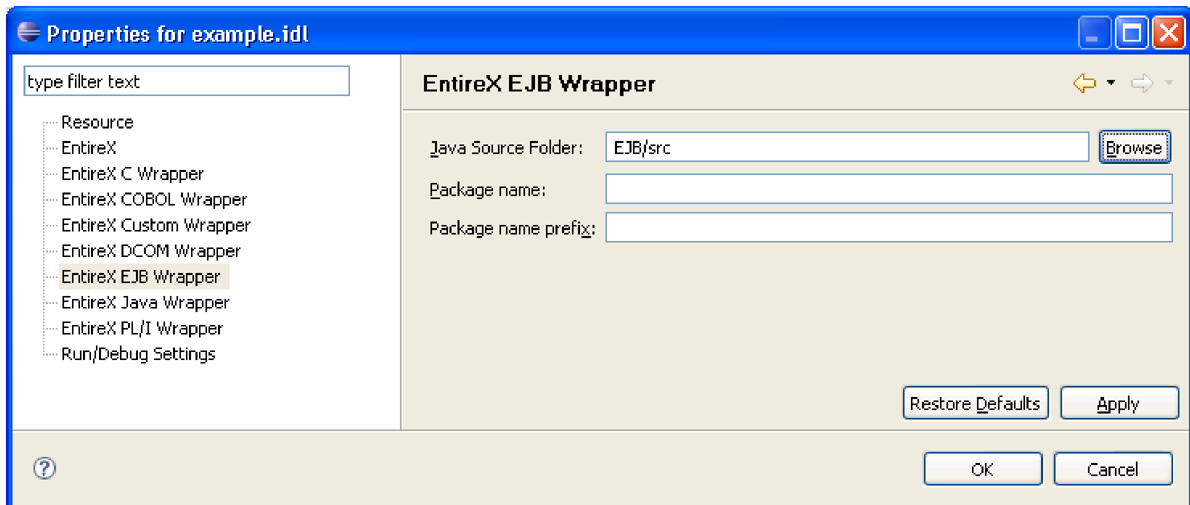
2 In the **Properties** window, choose the **EntireX** tab.

Here it is possible to modify the common default properties for the IDL file.



3 Choose the **EntireX Wrapper for Enterprise JavaBeans** tab.

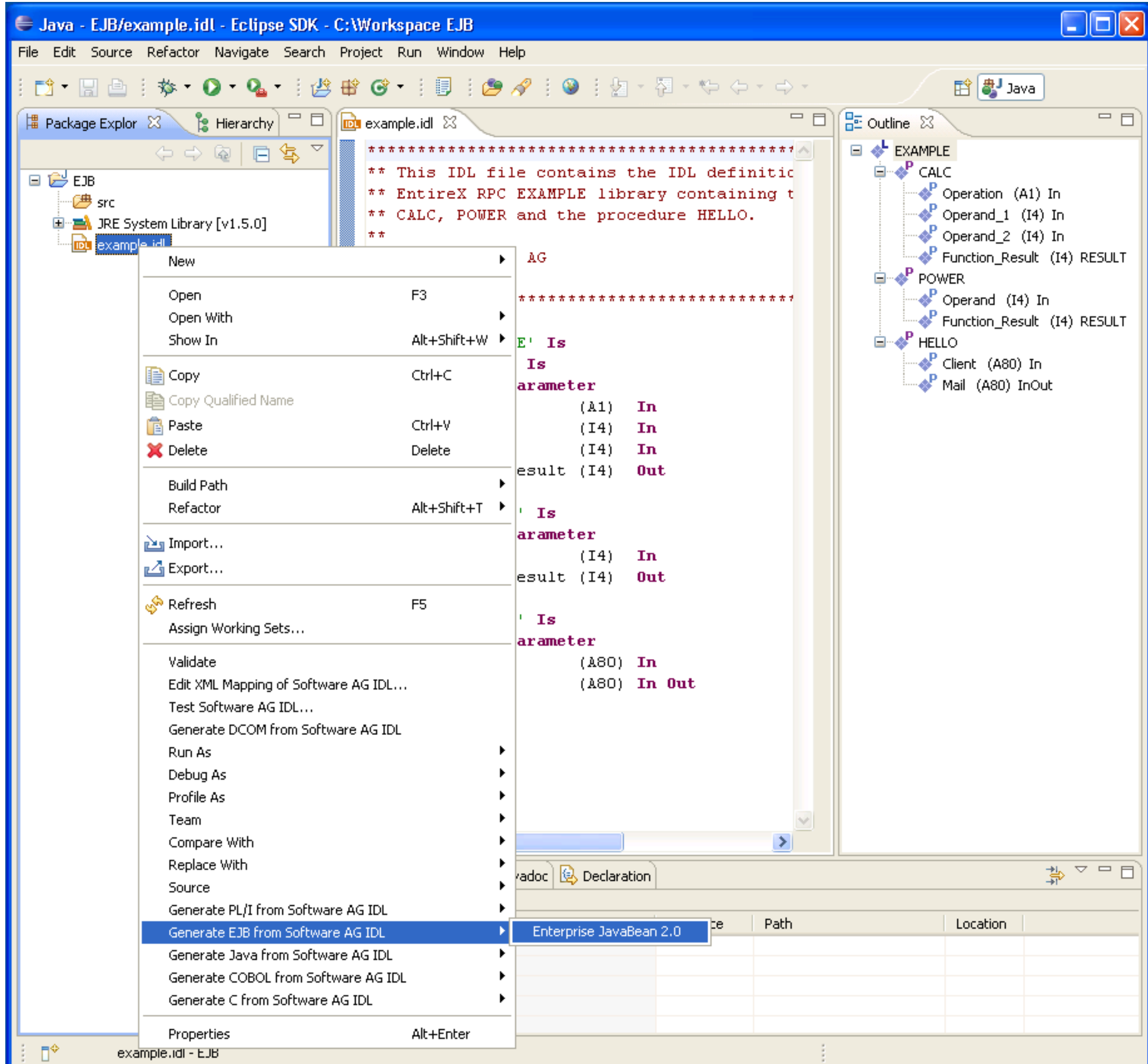
Here it is possible to modify the defaults (see table *Default Properties for the IDL File*, below).



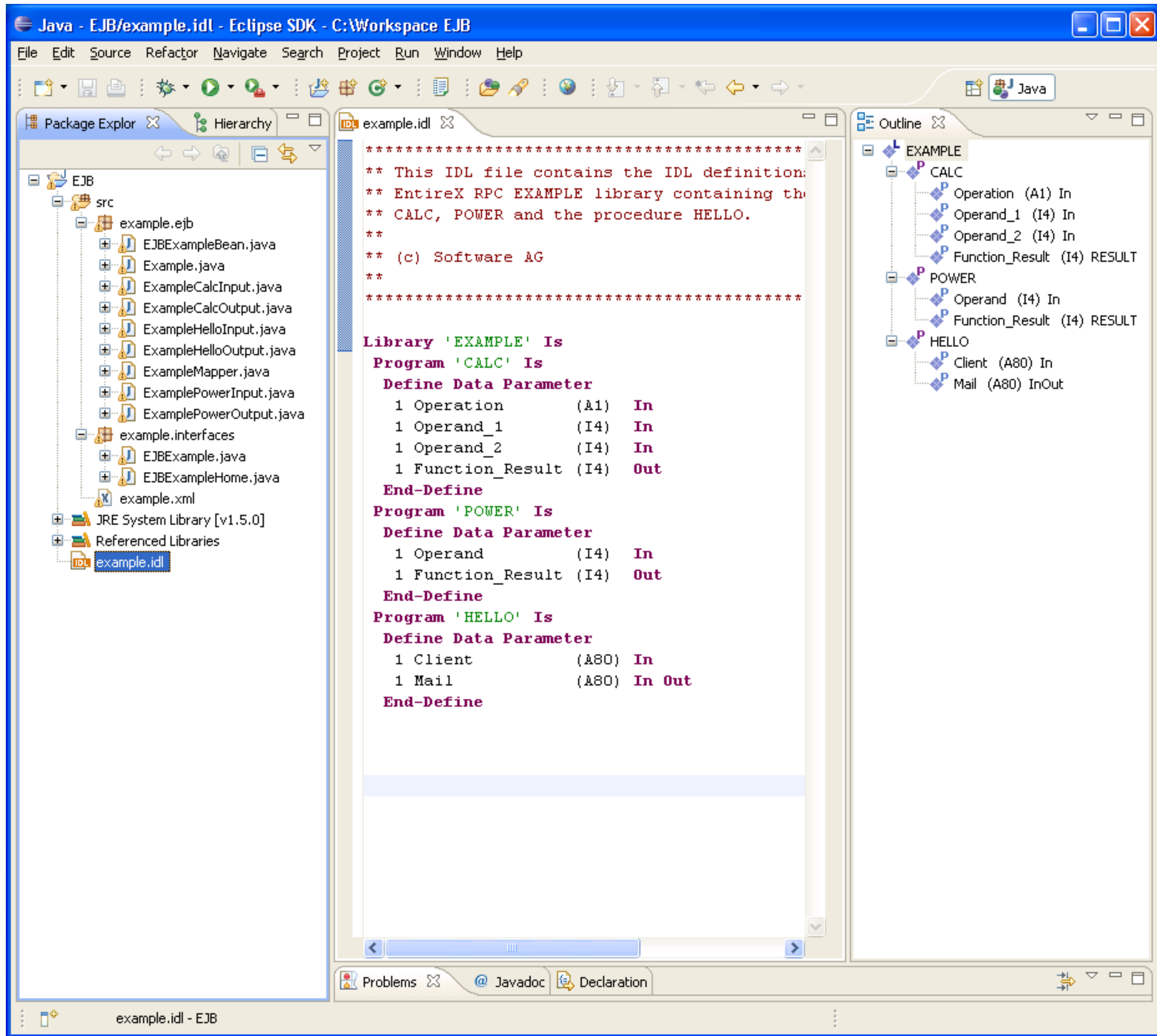
4 Confirm the entries with **OK**.

To generate the EJB sources

1 From the context menu, choose **Generate EJB from Software AG IDL > Enterprise JavaBeans 2.0**.



2 All generated EJB sources are in the subdirectory EJB of the home directory of the IDL file.


Java - EJB/example.idl - Eclipse SDK - C:\Workspace EJB
File Edit Source Refactor Navigate Search Project Run Window Help
Java
Package Explor
Hierarchy
EJB
src
example.ejb
EJBExampleBean.java
Example.java
ExampleCalcInput.java
ExampleCalcOutput.java
ExampleHelloInput.java
ExampleHelloOutput.java
ExampleMapper.java
ExamplePowerInput.java
ExamplePowerOutput.java
example.interfaces
EJBExample.java
EJBExampleHome.java
example.xml
JRE System Library [v1.5.0]
Referenced Libraries
example.idl
example.idl
****
** This IDL file contains the IDL definition
** EntireX RPC EXAMPLE library containing th
** CALC, POWER and the procedure HELLO.
**
** (c) Software AG
**
****
Library 'EXAMPLE' Is
Program 'CALC' Is
Define Data Parameter
1 Operation (A1) In
1 Operand_1 (I4) In
1 Operand_2 (I4) In
1 Function_Result (I4) Out
End-Define
Program 'POWER' Is
Define Data Parameter
1 Operand (I4) In
1 Function_Result (I4) Out
End-Define
Program 'HELLO' Is
Define Data Parameter
1 Client (A80) In
1 Mail (A80) In Out
End-Define
Outline
EXAMPLE
CALC
Operation (A1) In
Operand_1 (I4) In
Operand_2 (I4) In
Function_Result (I4) RESULT
POWER
Operand (I4) In
Function_Result (I4) RESULT
HELLO
Client (A80) In
Mail (A80) InOut
Problems
Javadoc
Declaration
example.idl - EJB

Generated Classes and Interfaces

The workbench generates the following interfaces and classes from the Software AG IDL file sources.

Home Interface

The generated home interfaces *EJB<libname>Home.java* contain the create methods. These methods can be called by an EJB client.

create()

```
EJB<libname> create()
```

Calling this method creates an EJB with the default settings.

create(String user)

```
EJB<libname> create(String user)
```

where: String user is the Broker user name

Calling this method creates an EJB with the default settings except for the user name.

create(String user, String password)

```
EJB<libname> create(String user,String password)
```

where: String user is the Broker user name and String password the Broker user password

Calling this method creates an EJB with the default settings except for the user name and the password.

Remote Interface

The generated remote interfaces *EJB<libname>.java* contain the customer methods. For each RPC program a customer method is generated with the following naming conventions.

```
<libname><program>Output <program> (<libname><program>Input input)
```

where: Output is the output object and Input is the input object

These methods can be called by an EJB client.

EJB Class

The generated EJB class *EJB<libname>Bean.java* extends *<libname>Mapper.java* class. Bean implements the EJB session bean and controls the EntireX RPC communication.

Mapper Class

The generated *<libname>Mapper.java* class extends the Java RPC client stub *<libname>.java*. It implements a method for each RPC Program. Methodname is *<program>*. The method has one parameter, an instance of *<libname><progname>*Input.class.The return value of the method is an instance of *<libname><progname>*Output.class.

Group Classes

The group classes are serializable representations of the inner classes of the groups of the RPC library, filename is *<libname><progname><innerclassname>.java*. The class contains all parameters of the group.

Struct Classes

The struct classes are serializable representations of the inner classes of the structs of RPC library, filename is *<libname><innerclassname>.java*. The class contains all parameters of the struct.

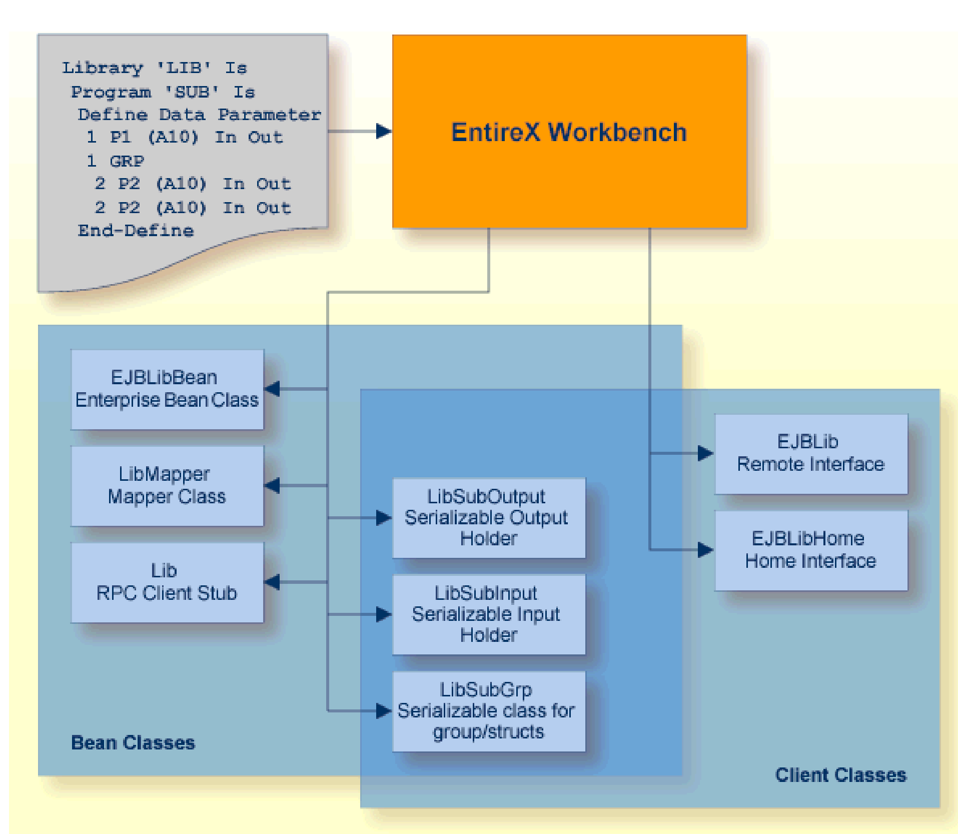
Input/Output Classes

These classes are serializable representations of all in, out, in/out parameters of RPC program. The filename of the input classes is *<libname><progname>Input.java*, it holds all in and in/out parameters of the program.The filename of the output classes is *<libname><progname>Output.java*, it holds all out and in/out parameters of the program.

Java RPC Client Stub

The generated Java RPC Stub `<libname>`*.java* extends com.softwareag.entirex.aci.RPCService class. It is the connecting link between the EJB and the EntireX server.

Example



Delivered Example

An example is delivered in directory *<drive>:\SoftwareAG\EntireX\examples\EJBWrapper* (Windows) or
/opt/softwareag/EntireX\examples\EJBWrapper (UNIX).

See the *README.TXT*.

3 Using the Wrapper for EJB in Command-line Mode

See *Using the EntireX Workbench in Command-line Mode* for the general command-line syntax. The table below shows the command-line options for the Wrapper for EJB.

The behavior of the generation in command-line mode depends on the IDL file:

- If the IDL file is part of the workspace, the generator generates the sources, controlled by the properties of the IDL file. The properties can be overwritten by the command-line options.
- If the IDL file is not part of the workspace but part of the file system, the generator creates the subdirectory EJB in the parent directory of the IDL file and generates the source files into this subdirectory. The generation can be controlled by the command-line options.

Task

Generate the EJB sources for the specified IDL file(s).

Command	Option	Default Value	Description
`-ejb:generate`	`-broker`	localhost:1971	Broker identification
	`-brokerpassword`	null	Broker user password.
	`-brokeruser`	EjbUserName	Broker user ID.
	`-compresslevel`	no	Compression level (No,Yes,0-9).
	`-encryption`	0	Encryption level (0,1,2).
	`-log`	null	Java ACI trace file.
	`-logicalbroker`	null	Logical Broker identifier.
	`-logicalservice`	null	Name of logical service.
	`-logonnaturalsecurity`	false	Logon to Natural.
	`-package`	IDL file name without extension	Package name for EJB classes.
	`-packageprefix`	null	Package name prefix.

Command	Option	Default Value	Description
	-rpcpassword	null	RPC user password.
	-rpcuser	null	RPC user ID.
	-security	false	Use EntireX Security.
	-server	RPC/SRV1/CALLNAT	Service name (class/server/service).
	-sourcefolder	<IDL file parent directory>/EJB	Folder of the generated classes.
	-trace	0	Java ACI trace level (0,1,2).
	-usecodepage	false	Send default codepage to the broker.
	-verbose	false	Enable EJB log information.

Example

```
<workbench> -ejb:generate /Demo/Example.idl -sourcefolder /Demo/src1
```

where `<workbench>` is a placeholder for the actual Workbench starter as described under *Using the EntireX Workbench in Command-line Mode*.

The name of the IDL file and the source folder include the project name. In the example, the project *Demo* is used.

If the first part of the IDL file name is not a project name in the current workspace, the IDL file name is used as a file name in the file system. Thus, the IDL files do not need to be part of an Eclipse project.

If the source folder does not exist in the workspace but the first part describing the project exists, the source folder is created.

If the IDL file is located outside the Eclipse workspace, the source folder is also a folder in the file system.

Status and processing messages are written to standard output (stdout), which is normally set to the executing shell window.

4 Software AG IDL to EJB Mapping

- Mapping IDL Data Types to Java Data Types 30
- Mapping Library Name and Alias 31
- Mapping Program Name and Alias 32
- Mapping Parameter Names 32
- Mapping Fixed and Unbounded Arrays 33
- Mapping Groups and Periodic Groups 33
- Mapping Structures 33
- Mapping the Direction Attributes IN, OUT and INOUT 34

Mapping IDL Data Types to Java Data Types

In the table below, the following metasymbols and informal terms are used for the IDL.

- The metasymbols [and] surround optional lexical entities.
- The informal term *number* (or *number* [.*number*]) is a sequence of numeric characters, for example 123.

Software AG IDL	Description	Java Data Types	Note
A*number*	Alphanumeric	String	1, 3
AV	Alphanumeric variable length	String	
AV[*number*]	Alphanumeric variable length with maximum length	String	1
B *number*	Binary	byte[]	1, 6
BV	Binary variable length	byte[]	
BV[*number*]	Binary variable length with maximum length	byte[]	1
D	Date	java.util.Date	5
F4	Floating point (small)	float	2
F8	Floating point (large)	double	2
I1	Integer (small)	byte	
I2	Integer (medium)	short	
I4	Integer (large)	int	
K *number*	Kanji	String	1
KV	Kanji variable length	String	
KV[*number*]	Kanji variable length with maximum length	String	1
L	Logical	boolean	
N *number* [.*number*]	Unpacked decimal	java.math.BigDecimal	4
NU *number* [.*number*]	Unpacked decimal unsigned	java.math.BigDecimal	4
P *number* [.*number*]	Packed decimal	java.math.BigDecimal	4
PU *number* [.*number*]	Packed decimal unsigned	java.math.BigDecimal	4
T	Time	java.util.Date	5
U *number*	Unicode	String	7
UV	Unicode variable length	String	7
UV *number*	Unicode variable length with maximum length	String	7

Notes:

1. The field length is given in bytes.

2. If floating-point data types are used, rounding errors can occur. Therefore, the values of sender and receiver might differ slightly.
3. If you use the value null (null pointer) as an input parameter (for `IN` and `INOUT` parameters) for type A, a blank string will be used.
4. If you use the value null (null pointer) as an input parameter (for `IN` and `INOUT` parameters) for types N/P, the value 0 (or 0.0) will be used.
5. If you use the value null (null pointer) as an input parameter (for `IN` and `INOUT` parameters) for types D/T, the current date/time will be used. You change this with the property `entirex.marshal.date`. Setting `entirex.marshal.date=null` will map the value null to the invalid date 0000-01-01 of the RPC marshalling. This is the invalid date value in Natural, too. With this setting the invalid date as an output parameter will be mapped to null. The default is to map the invalid date to 0001-01-01.
6. If you use the value null (null pointer) as an input parameter (for `IN` and `INOUT` parameters) for type B, all binary values will be set to zero.
7. The length is given in 2-byte Unicode code units following the Unicode standard UTF-16. The maximum length is 805306367 code units.

Please note also hints and restrictions on the Software AG IDL data types valid for all programming language bindings. See *IDL Data Types* under *Software AG IDL File* in the IDL Editor documentation.

Mapping Library Name and Alias

The library name in the IDL file is mapped to the class name of the generated bean. See `library-definition` under *Software AG IDL Grammar* in the *IDL Editor* documentation. For the bean, the names of the classes have the format *`library-name`*`Bean`.

The special characters '#' and '-' in the library name are replaced by the character '_'.

If there is an alias for the library name in the `library-definition` under *Software AG IDL Grammar* in the *IDL Editor* documentation, this alias is used “as is” to form the bean class name. Therefore, this alias must be a valid Java class name.

Example:

- library name `Hu#G-O` is converted to `EJBHu_g_oBean.class`

The library name is sent - without changes - to the server. The library alias is never sent to the server.

In the RPC server the library name sent may be used to locate the target server.

Mapping Program Name and Alias

The program name in the IDL file is mapped to method names within the generated Enterprise JavaBeans. See `program-definition` under *Software AG IDL Grammar* in the *IDL Editor* documentation. To adhere to Java naming conventions, the program name is converted to lowercase.

The special characters '#' and '-' in the program name are replaced by the character '_'.

If there is an alias for the program name in the `program-definition` under *Software AG IDL Grammar* in the *IDL Editor* documentation, this alias is used "as is" for the method name. Therefore, this alias must be a valid Java method name.

The program name is converted to uppercase before it is sent to the server. The program alias is never sent to the server.

The program name sent to the RPC server is used to locate the target server.

Mapping Parameter Names

The parameter names are mapped to fields inside the serializable input and output classes (see ***Mapping the Direction Attribute IN, OUT, and INOUT***). The name of the input class is made up of the library name and the program name `<library-name><program-name>Input` and for the output class is `<library-name><program-name>Output`.

Example:

```
public class LibPgmInput implements Serializable {
 public String myInputString;
 public String myOutInString;
}
```

or

```
public class LibPgmOutput implements Serializable {
 public String myOutputString;
 public String myOutInString;
}
```

Mapping Fixed and Unbounded Arrays

Arrays in the IDL file are mapped to Java arrays. If an array value does not have the correct number of dimensions or elements, this will result in a `NullPointerException` or an `ArrayIndexOutOfBoundsException`. If you use the value null (null pointer) as an input parameter (for `IN` and `INOUT` parameters), an array will be instantiated.

Mapping Groups and Periodic Groups

Groups are mapped to serializable classes. The class name is the *<library name><program name><parameter name>*,

For example, the following Software AG IDL

```
Library 'Lib1130' Is
 Program 'X201G0' Is
  Define Data Parameter
   1 MyGroup
    2 MyLong (I4)
    2 MyFloat (F4)
   1 MyGroupAsString (A253)
   1 function_result (I4) Out
  End-Define
```

is mapped to the class

```
public class Lib1130X201g0MyGroup implements Serializable {
 public int mylong;
 public float myfloat;
}
```

Mapping Structures

Structures are mapped to serializable classes. The class name is the *<library name><struct name>*.

Mapping the Direction Attributes IN, OUT and INOUT

IDL syntax allows you to define parameters as `IN` parameters, `OUT` parameters, or `INOUT` parameters (the default). This specification of the direction is reflected in the generated Enterprise JavaBeans as follows:

- IN parameters are sent from the RPC client to the RPC server. `IN` parameters are collected in the serializable input class (see ***Mapping Parameter Names***).
- OUT parameters are sent from the RPC server to the RPC client. `OUT` parameters are collected in the serializable output class (see ***Mapping Parameter Names***).
- INOUT parameters are sent from the RPC client to the RPC server and then back to the RPC client. `INOUT` are collected into both classes the serializable input class as well as into the serializable output class. (see ***Mapping Parameter Names***).

Note: Only the direction information of the top-level fields (level 1) is relevant. Group fields always inherit the specification from their parent. Any different specification is ignored.

See the `attribute-list` under *Software AG IDL Grammar* in the *IDL Editor* documentation for the syntax on how to describe attributes in the Software AG IDL file and refer to the `direction` attribute.

5 Writing Applications with the Wrapper for EJB

- Programming a Client Application 36
- Compiling and Running the Client Application 37

Programming a Client Application

To program a client which uses an enterprise bean, perform the following tasks:

Locate the Home Interface

- Create the initial context

```
Context initial=new InitialContext();
```

- Get the object bound to the JNDI name EJBExample

```
Context myEnv=(Context) initial.lookup("EJBExample");
```

- Narrow the reference to an EJBExampleHome object

```
EJBExampleHome ↩
home=(EJBExampleHome)PortableRemoteObject.narrow(objref,EJBExampleHome.class)
```

Create an Enterprise Java Bean Instance

- Declare the bean object

```
EJBExample ejbExample=null;
```

- Create the bean object with default settings

```
ejbExample=home.create();
```

- Or create the bean object with default settings except for the user name

```
ejbExample=home.create("myUserName");
```

- Or create the bean object with default settings except for the user name and password

```
ejbExample=home.create("myUserName","myPassword");
```

Invoke the Mapped EntireX RPC method

- Create the input and output objects

```
ExampleSquareInput input=new ExampleSquareInput();
ExampleSquareOutput output=new ExampleSquareOutput();
```

- Set the input value

```
input.operand=9;
```

- Show the input value

```
System.out.println( "square (input) >>> "+intput.operand );
```

- Call the EntireX RPC method

```
output = ejbExample.square(input);
```

- Show the output value

```
System.out.println( "square (input) >>> "+output.result );
```

- Remove the bean object

```
ejbExample.remove();
```

Compiling and Running the Client Application

- **Compile the client with the client JAR**
  It will be packed by the Application Server or by the Ant script.
- **Get the *build.xml* file from the example directory**
  Set antfile to your current XML file.

  Replace Client.java by your client class name.

To run the client

- Enter

```
ant jboss_run_client
```

6 Controlling Applications - EntireX Wrapper for Enterprise JavaBeans

- Environment Entries to Control EJB 40
- Using Security/Encryption 41
- Using Natural Security 41
- Using Compression 42
- Using Internationalization with Wrapper for EJB 42
- Tracing 43
- Deployment with an Application Server 43

Environment Entries to Control EJB

To define the behavior of the EJB

- Use the following environment entries

Entry Name	Default Value	Description
Broker	Localhost:1971	Broker ID
Server	RPC/SRV1/CALLNAT	Service name (class/server/service)
User	EjbUserName	Broker user ID
Password	null	Broker user password
Compression Level	no	Compression level (No,Yes,0-9)
Codepage	false	Use codepage
Security	false	Use EntireX security
Encryption	0	Encryption level (0,1,2)
Natural Security	false	Use Natural Security
RPC User	null	RPC user ID
RPC Password	null	RPC user password
Verbose	false	Bean log information
Trace	0	Java ACI trace level (0,1,2)
Logfile	null	Java ACI trace file

Using Security/Encryption

To use EntireX Security

- Set entry name `Security` to "true", User to the current user and Password to the current password.

With these settings the Broker checks user and password.

To use EntireX Security with encryption level broker

- Set entry name `Security` to "true", Encryption equals 1, `User` to the current user and `Password` to the current password.

With these settings the Broker checks user and password, the data of send and receive calls are encrypted from the application server to the Broker.

To use EntireX Security with encryption level target

- Set entry name `Security` to "true", Encryption equals 2, `User` to the current user and `Password` to the current password.

With these settings the Broker checks user and password, and the data of send and receive calls are encrypted from the application server to the RPC server.

Using Natural Security

To use Natural Security

- Set entry name `Natural Security` to "true", `User` to the current Natural user and `Password` to the current Natural password.

With these settings the Broker checks Natural user and Natural password.

If you want to use EntireX Security and Natural Security and the Natural Security user name is not the same as the EntireX Security user name, set entry name `RPC User` to the current Natural user. If the Natural Security password is the same as the EntireX Security password, set entry name `RPC User` to the current Natural user.

Using Compression

To use EntireX Compression

- Set entry name `Compression level` to one of the following values:

Value	Description	Data Type
9	Best Compression	string
1	Best Speed	string
Y(es) or 6	Default Compression	string
8	Deflated	string
N(o) or 0	No Compression	string

Using Internationalization with Wrapper for EJB

It is assumed that you have read the document *Internationalization with EntireX* and are familiar with the various internationalization approaches described there.

EntireX Java components use the codepage configured for the Java virtual machine (JVM) to convert the Unicode (UTF-16) representation within Java to the multibyte or single-byte encoding sent to or received from the broker by default. This codepage is also transferred as part of the locale string to tell the broker the encoding of the data if communicating with a broker version 7.2.x and above.

To change the default, see your JVM documentation. On some JVM implementations, it can be changed with the `file.encoding` property. On some UNIX implementations, it can be changed with the LANG environment variable.

Which encodings are valid depends on the version of your JVM. For a list of valid encodings, see *Supported Encodings* in your Java documentation. The encoding must also be a supported codepage of the broker, depending on the internationalization approach.

With the entry name `Codepage` you can

- force a locale string to be sent if communicating with Broker version 7.1.x and below. Set entry name `Codepage` to "true" for this purpose.
- not use a codepage other than the default encoding of the JVM.

Tracing

To switch on bean tracing

- Set entry name `Verbose` to "true".

Some EJB-relevant information is written to the standard output of the application server.

To switch on Java ACI tracing

- Set entry name `Trace` to "0", "1", "2" or "3".

  The output will be written to standard output.

Trace Level	Description
0	No trace output
1	Trace all Broker calls and other major actions
2	Dump the send-and-receive buffer
3	Dump the buffers sent to the Broker and received from the Broker

To write the trace information to a file

- Set the entry name `Logfile` to the corresponding file.

Deployment with an Application Server

General Information

Note: Version numbers to third-party products are listed centrally in the section *EntireX Prerequisites*. In the examples, notation such as *nn* or *nnn* refers to the two- or three-digit version numbers.

Make sure that the application server can access the EntireX Java Runtime.

For the compilation of the generated sources and the generation of the deployment descriptors we use Ant and XDoclet.

Load, install, and verify Ant. See *<cd-root>/share/3rdparty/ant nnn.zip* or ***http://ant.apache.org/***.

Load, install, and verify XDoclet. See *<cd-root>/share/3rdparty/xdoc* nnn*.zip* or ***http://sourceforge.net/projects/xdoclet/***.

Windows users:
If your path names or file names contain blanks, you have to use the short name notation without blanks, e.g.: Use *C:\PROGRA~1\XYZ* instead of *C:\Program Files\xyz*

Please refer to the vendor for more details about the application server used.

Deployment with a WebSphere Application Server under UNIX

Note: The following description has been verified with the delivered EJB example and WebSphere 5 on RedHat Linux Advanced Server 2.1 for x86 and might be different for other WebSphere releases.

The following steps are described in this section:

- Compiling the EJB Example
- Generating Code for Deployment
- Creating an Application Client
- Creating an Application
- EJB Deployment

Compiling the EJB Example

To compile the example

1. You can use the *build.xml* file as a template, which is provided in directory *examples/EJBWrapper*.
2. Change the assignment *antfile="example.xml"* according to your own XML file name.
3. Adapt the basedir and the following properties in the *build.xml* file:

```
   <project name="Example" default="all" ↩
basedir="/SAG/exx/vnnn/examples/ejb_wrapper">
      <target name="init">
       <property name="exx.classes" value="/SAG/exx/vnnn/classes/entirex.jar"/>
        <property name="ant.classes" value="/SAG/antnnn/lib/ant.jar"/>
       <property name="xdoclet.classes" value="/SAG/xdocnnn/lib/xdoclet.jar"/>
      </target>
      <target name="ibm_init" depends="init">
        <property name="java.classes" value="/opt/jdknnn_06/lib/tools.jar"/>
        <property name="j2ee.classes" value="/opt/j2sdkeennn/lib/j2ee.jar"/>
      </target>
```

4. Compile the bean components:

```
ant -buildfile build.xml ibm_compile
```

5 Compile the client main program:

```
ant -buildfile build.xml ibm_compile_client
```

Generating Code for Deployment

To generate code for deployment

1 Start the Assembly tool of WebSphere:

```
/opt/WebSphere/AppServer/bin/assembly.sh
```

2 From the **New** tab choose EJB Module.

In the left frame of the screen the types of beans belonging to the module are listed (Session Beans, Entity Beans, Message Driven Beans, etc.), while the right frame shows the properties.

3 Create a new deployment descriptor for the bean by selecting **Session Beans** and choosing **New**.

The window **New Session Bean** will be displayed.

4 In the tab **General** enter the EJBName, the EJB class, and in the Remote section Home and Interface:

```
       EJB name:  EJBExampleBean
       EJB class: example.ejb.EJBExampleBean
       Remote
       Home:      example.interfaces.EJBExampleHome
       Interface: example.interfaces.EJBExample
```

5 In the tab **Advanced** choose session type Stateful.

6 In the tab **Bindings** set the JNDI Name

```
JNDI name: EJBExample
```

7 Confirm with **OK**.

8 On the left screen choose **Add Files**.

9 Add all class files created by the *EntireX Workbench* EJB function and compile.

- Browse and select *exx/vnnn/examples/ejb_wrapper* and Add Client.class.
- Browse to *exx/vnnn/examples/ejb_wrapper* and choose **EJB**.
- Select **example** in the right window and choose **Add.**

(it is important that the selected files have a path name such as *example/ejb/file.class*).

10 Add all class files of *entirex.jar*

- Browse to *exx/vnnn/classes* and choose *entirex.jar*.

  Select **com**, choose **Add** and **OK** (do not add the META-INF).
- Save the archive EJBExample.jar.

11 From the **File** menu, choose **Generate Code** for deployment... and then **Generate Now**.

A file *Deployed_EJBExample.jar* will be generated.

12 Close the bean module.

Creating an Application Client

To create an application client

1. Create a new application client. From the **File** menu, choose **New** and **Application Client**.
2. Change the Display name and complete the text field **Client** in the Main class.
3. Select **EJB References** and choose **New**.
4. In the tab **General** enter the Bean name *ejb/MyExample*.
5. Obtain the environment naming context of the application client.
6. Enter the Home and Remote interface:

```
Name:   ejb/MyExample
Home:   example.interfaces.EJBExampleHome
Remote: example.interfaces.EJBExample
```

7. In the tab **Bindings** enter the JNDI name.

```
JNDI name: EJBExample
```

8. Save the application client (*ApplicationClient.jar*).

Creating an Application

To create a new application

1. From the **File** menu, choose **New** and choose **Application**.
2. Change the Display name.
3. Select Application Clients and import *ApplicationClient.jar*.
4. Select EJB Module and import *Deployed_EJBExample.jar*.
5. Save the application as *Application.ear*.

EJB Deployment

To deploy the EJB

1. Open an Administration console in a Browser (http://localhost:9090/admin).
2. From the **File** menu, choose **Applications** and choose **Install New Application** and enter the path to your application (enter path to the file *Application.ear*).
3. Choose **Next**.
4. Specify the prefix you have already chosen for the client application (*etb/MyExample*) and choose **Do not override existing bindings**.
5. Choose **Next**.
6. Choose **Enable class reloading** and **Next**.
7. Verify the JNDI name and choose **Next**.
8. Choose EJB Module, **Next** and then **Finish**.
9. Save the configuration with **Save to Master Configuration**.
10. Choose **Enterprise Applications** and start the application.