Deployment Manual

SAP J2EE Engine 6.20

Contents

About This Manual 4
Target Audience 4
Structure 4
Deployment Tasks 5
Overview 6
Generate J2EE Components 7
Generate J2EE Components Using Deploy Tool 7
Generate J2EE Components Using Script File 7
Generate J2EE Components Using Own Java Classes 12
Application Assembling 18
Assembling Using Deploy Tool 18
Assembling Using Script Files 18
Assembling Using Own Java Classes 22
Deployment 26
Deployment Using Deploy Tool 26
Deployment Using Script Files 26
Deployment Using Shell Commands 38
Deployment Using Visual Administrator 38
Deployment Using Own Java Classes 38
Deploy Tool 47
Overview 48
Deploy Tool Components 49
Main Frame 50
Keyboard Shortcuts 52
Generate J2EE Components 54
Add EJB Group 54
Add EJB 60
Add Web 69
Add Servlet|JSP 83
Add Java Client Archive 86
Remove 93
Make Archives 93
Make All Archives 93
View Log File 93
Application Assembling 94

Add Archive 94
Remove 94
Edit the Archive Descriptors 94
Edit the J2EE EAR 96
Additional Options 99
Deployment 100
Storage.xml 105
Getting Started with Deploy Tool 109
Introduction 110
Step 1: Start the Deploy Tool 111
Step 2: Create a New Project File 112
Step 3: Add an EJB Group 114
Step 4: Set the Project Classpath 116
Step 5: Add the EJB 117
Step 6: Add a WEB Archive Group 120
Step 7: Add a Directory to the WEB Archive 122
Step 8: Add Files to the WEB Archive 124
Step 9: Add the Servlet 125
Step 10: Set Servlet Mappings 127
Step 11: Set EJB References 128
Step 12: Make the Archives 129
Step 13: Make the EAR 131
Step 14: Connect to SAP J2EE Engine 132
Step 15: Deploy 133
Run the Example 135
Troubleshooting and Known Bugs 136
Troubleshooting and Known Bugs 137

About This Manual

This manual concerns the deployment of J2EE applications on SAP J2EE Engine. It describes the deployment process from the defining of the deployment task to the application deployment. All deployment scenarios are described in detail. An easy-to-use Deploy Tool is provided with SAP J2EE Engine. Its functions and usage are described in the *Deploy Tool* section.

Target Audience

Deployment Manual is addressed to application developers and installers with strong knowledge of Java programming language and J2EE™ Specification. This document is aimed at administrators and developers familiar with the specifics of the application to be deployed.

Structure

The content of this manual is organized into several subsections, as described below:

- Deployment Tasks – covers the basic points of each deployment task and the different alternatives according to the available application components
- Application Assembling – describes the assembling of all necessary application components – EJBs, JSPs, Servlets, and so on, the different deployment descriptors; assembling using different tools; and the usage of another IDE for the application assembling
- Deployment – explains the deploy time configuration tasks and shows how to perform them using SAP J2EE Engine Visual Administrator, Java classes or shell commands (or both). Several examples are provided.
- Deploy Tool – describes SAP J2EE Engine Deploy Tool and its functions. Deploy Tool enables easy application assembly and deployment. This section describes the Deploy Tool interface and its usage.
- Getting Started with Deploy Tool – contains a Deploy Tool Tutorial that describes the deployment of an example application step by step

Chapter 1

Deployment Tasks

- Overview
- Generate J2EE Components
- Application Assembling
- Deployment

Overview

Deployment is the process of installing applications on the server side to be used by clients. SAP J2EE Engine, being a fully certified J2EE compliant application server, supplies deployment of J2EE enterprise application archives (EAR files). The deployment procedure can start at different levels according to the existing components and deployment tasks. Typically, the deployment process is separated into three steps. There are various alternative ways to perform each of these steps, as shown below:

Task	Ways to perform
Generate J2EE Components (create JAR and WAR archive files out of Java classes, EJBs, JSP, Servlets, and so on.)	• SAP J2EE Engine Deploy Tool • SAP J2EE Engine script file • SAP J2EE Engine Java API • Third party IDE (for example Borland JBuilder™ and so on.)
Application Assembling (create EAR out of JARs and WARs)	• SAP J2EE Engine Deploy Tool • SAP J2EE Engine script file • SAP J2EE Engine Java API
Deployment (install the application EAR on SAP J2EE Engine)	• SAP J2EE Engine Deploy Tool • SAP J2EE Engine script file • SAP J2EE Engine Java API • SAP J2EE Engine Shell command • SAP J2EE Engine Visual Administrator

SAP J2EE Engine Deploy Tool is the best tool to perform all tasks. It is described in the *Deploy Tool* section of this document. You can use other ways, if convenient.

Everyone can use the Deploy Tool to generate and assemble J2EE components. By default, only the administrator of SAP J2EE Engine can deploy applications.

Note: If you write the XML files for SAP J2EE Engine script file manually, be careful to save them as plain text. It is critical that the XML file does not contain non-text characters. If the file contains formatting symbols the script files do not work properly. For example, write XML files on a text editor (such as Notepad) and save them using ANSI encoding.

To manually write archive descriptors use J2EE™ Specification v1.2 (Enterprise JavaBeans™ Specification v1.1, Java™ Servlet Specification v2.2 and JavaServer Pages™ Specification 1.1).

SAP J2EE Engine supports deployment of Java™ Servlet Specification v2.2 compliant servlets.

Generate J2EE Components

To deploy an application on SAP J2EE Engine, its components must be developed. They are J2EE components and must be distributed in Java archive files – JAR, WAR and client JAR files. Therefore, J2EE components must be generated. EJB files must be grouped in JAR files; Servlets and JSPs in WAR files. Other files can be added, according to the application. At this stage, the relations between the components are defined. Typically, you can use three methods to generate J2EE components – Deploy Tool, script files and your own Java classes.

Generate J2EE Components Using Deploy Tool

SAP J2EE Engine Deploy Tool performs this task using its GUI environment. For further information, refer to the *Deploy Tool* section of this document.

Generate J2EE Components Using Script File

To build your components into JAR and WAR files, you can use the *J2EEgenerator* script file, located in the *<SAPj2eeEngine_install_dir>/deploying* directory.

When executing the script file, you must specify the following arguments on the command line:

- `<XML-file>` – the path to the XML file that the *J2EEgenerator* script file uses to create the archives. The structure of this XML file is described below.
- `[LOG-file]` – the file where log messages are stored. If this argument is not specified, a file named *j2ee_generator_log.txt* is created in the *<SAPj2eeEngine_install_dir>/deploying* directory by default and the log messages are stored in it.

To execute the script file, enter the following on the command line:

```
J2EEgenerator <XML-file> [LOG-file]
```

When the *J2EEgenerator* script file is executed, validity verification for each tag in the XML file is performed and in case of incorrect content an exception is thrown. The procedure terminates, and the reason for interruption is written to the log file.

The XML file used by the *J2EEgenerator* script file must have the following document type definition:

```
<!DOCTYPE j2ee-components-generator [
<!ELEMENT j2ee-components-generator (project-dir?, archive*)>
<!ELEMENT project-dir (#PCDATA)>
<!ELEMENT archive (file-name, archive-descriptor, files*,
directories*, additional-classpath)>
<!ELEMENT file-name (#PCDATA)>
<!ELEMENT archive-descriptor (file-location)>
<!ATTLIST archive-descriptor type CDATA #REQUIRED>
<!ELEMENT file-location (#PCDATA)>
<!ELEMENT files (entry-name, file-path)>
<!ELEMENT entry-name (#PCDATA)>
<!ELEMENT file-path (#PCDATA)>
<!ELEMENT directories (dir-path, extension-mapping*, included,
```

```
excluded-files?)>
<!ELEMENT dir-path (#PCDATA)>
<!ELEMENT included (#PCDATA)>
<!ELEMENT extension-mapping (ext, mapping)>
<!ELEMENT ext (#PCDATA)>
<!ELEMENT mapping (#PCDATA)>
<!ELEMENT excluded-files (#PCDATA)>
<!ELEMENT additional-classpath (#PCDATA)>
]>
```

All fields are explained below:

Note: The DOCTYPE field must be specified properly for the XML file to be parsed.

```
<!--
J2ee-components-generator element is the root element. This
element is obligatory and the name has to be j2ee-components-
generator.
-->
<!ELEMENT j2ee-components-generator (project-dir?, archive*)>

<!--
Project-dir element specifies the project directory, where project
files are located. This is a main element for the application
components described with relative file path. If this element is
not specified the location of the application components have to
be specified by their full file path.
Used in: j2ee-components-generator
-->
<!ELEMENT project-dir (#PCDATA)>

<!--
Archive element describes the archive to be created. You can use
J2EEgenerator script file to generate more than one J2EE
components simultaneously, for example – JAR, WAR, client
archives.
Used in: j2ee-components-generator
-->
<!ELEMENT archive (file-name, archive-descriptor, files*,
directories*, additional-classpath)>

<!--
File-name element specifies the relative or full path to the
output archive file to be created by the J2EEgenerator script
file.
Used in: archive
-->
<!ELEMENT file-name (#PCDATA)>

<!--
Archive-descriptor element specifies the relative or full path to
the archive deployment descriptor, an XML file. This file must be
created in advance.
Used in: archive
-->
<!ELEMENT archive-descriptor (file-location)>

<!--
Archive-descriptor attribute specifies the type of the J2EE
archive element that deployment descriptor describes – EJB JAR,
WAR or client JAR files. Possible values are “EJB,” “WEB” or
“JAVA” respectively.
-->
<!ATTLIST archive-descriptor type CDATA #REQUIRED>
```

```
<!--
File-location element specifies the relative or full path to the
XML file, describing the archive.
Used in: archive-descriptor
-->
<!ELEMENT file-location (#PCDATA)>

<!--
Files element specifies the entry name and the relative or full
path to the additional to the archive files. This tag is required
if helper classes or other additional files (or both) exist in the
project. The tag is not used to specify EJB classes in a JAR, as
they are described in the deployment descriptor. The Servlet
classes and JSP files in a War must be specified. The MANIFEST.MF
files of the components must be specified, too.
Used in: archive
-->
<!ELEMENT files (entry-name, file-path)>

<!--
Entry-name specifies the entry name of the additional file(s) in
the archive. This element can be used with mapping purposes. For
example, MANIFEST.MF files must be added with META-INF\MANIFEST.MF
entry-name.
Used in: files
-->
<!ELEMENT entry-name (#PCDATA)>

<!--
File-path element specifies the relative or full path to the
additional file(s).
Used in: files
-->
<!ELEMENT file-path (#PCDATA)>

<!--
Directories element specifies a directory and its sub directories
to be added to the archive.
Used in: archive
-->
<!ELEMENT directories (dir-path, extension-mapping*, included,
excluded-files?)>

<!--
Dir-path element specifies the path to the directories, that will
be added to the archive.
Used in: directories
-->
<!ELEMENT dir-path (#PCDATA)>

<!--
Included element specifies whether the directory is added to the
archive. The possible values are “YES” or “NO.” This element is
obligatory for a directory to be added or not respectively.
Used in: directories
-->
<!ELEMENT included (#PCDATA)>

<!--
Extension-mapping element describes a list of all mappings and
filters defined for the files in the directory.
Used in: directories
-->
<!ELEMENT extension-mapping (ext, mapping)>
```

```
<!--
Ext element specifies the extension by which all files are
filtered, from the directory to be added. The possible values are
mark wit a full stop at the beginning. For example “.class” or
“.txt,” and so on.
Used in: extension-mapping
-->
<!ELEMENT ext (#PCDATA)>

<!--
Mapping element specifies the directory where the filtered files
are added to the archive. For example WEB-INF.
Used in: extension-mapping
-->
<!ELEMENT mapping (#PCDATA)>

<!--
Exclude-files element specifies a list of extensions of files to
be excluded from the directories to be added to the archive. For
example, .xml;.mf;
Used in: directories
-->
<!ELEMENT excluded-files (#PCDATA)>

<!--
Additional-classpath element specifies a list of additional
classpaths used for loading the EJB. The different classpaths are
divided by semicolon (;).
Used in: archive
-->
<!ELEMENT additional-classpath (#PCDATA)>
```

Example:

To build the example components from *<SAPj2eeEngine_install_dir>/deploying/deploy_example* directory into JAR and WAR files, use the script file *<SAPj2eeEngine_install_dir>/deploying/J2EEgenerator*.

To execute the script file, enter the following on the command line:

```
J2EEgenerator <XML-file> [LOG-file]
```

Where:

- `<XML-file>` – the path to the XML file *<SAPj2eeEngine_install_dir>/deploying/deploy_example/J2EEgen.xml*
- `[LOG-file]` – the file where log messages are stored. If this argument is not specified, a file named *j2ee_generator_log.txt* is created in the *<SAPj2eeEngine_install_dir>/deploying* directory by default.

An example for XML file used by the *J2EEgenerator* script file follows:

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE j2ee-components-generator [
<!ELEMENT j2ee-components-generator (project-dir?, archive*)>
<!ELEMENT project-dir (#PCDATA)>
<!ELEMENT archive (file-name, archive-descriptor, files*,
directories*, additional-classpath)>
<!ELEMENT file-name (#PCDATA)>
<!ELEMENT archive-descriptor (file-location)>
<!ATTLIST archive-descriptor type CDATA #REQUIRED>
<!ELEMENT file-location (#PCDATA)>
```

```
<!ELEMENT files (entry-name, file-path)>
<!ELEMENT entry-name (#PCDATA)>
<!ELEMENT file-path (#PCDATA)>
<!ELEMENT directories (dir-path, extension-mapping*, included,
excluded-files?)>
<!ELEMENT dir-path (#PCDATA)>
<!ELEMENT included (#PCDATA)>
<!ELEMENT extension-mapping (ext, mapping)>
<!ELEMENT ext (#PCDATA)>
<!ELEMENT mapping (#PCDATA)>
<!ELEMENT excluded-files (#PCDATA)>
<!ELEMENT additional-classpath (#PCDATA)>
]>

  <j2ee-components-generator>
    <project-dir>
      C:\J2EEngine\deploying
    </project-dir>
    <archive>
      <file-name>
        \one_example\jar.jar
      </file-name>
      <archive-descriptor type="EJB">
        <file-location>
          \one_example\jar_jar_DD.xml
        </file-location>
      </archive-descriptor>
      <files>
        <entry-name>
          META-INF/MANIFEST.MF
        </entry-name>
        <file-path>
          \one_example\jar_jar.mf
        </file-path>
      </files>
      <additional-classpath>
        C:\J2EEngine\deploying\deploy_example;
      </additional-classpath>
    </archive>
    <archive>
      <file-name>
        \one_example\Client.jar
      </file-name>
      <archive-descriptor type="JAVA">
        <file-location>
          \one_example\Client_jar_DD.xml
        </file-location>
      </archive-descriptor>
      <files>
        <entry-name>
          META-INF/MANIFEST.MF
        </entry-name>
        <file-path>
          \one_example\Client_jar.mf
        </file-path>
      </files>
      <additional-classpath>
        C:\J2EEngine\deploying\deploy_example;
      </additional-classpath>
    </archive>
    <archive>
      <file-name>
        \one_example\War.war
      </file-name>
      <archive-descriptor type="WEB">
        <file-location>
```

```
        \one_example\War_war_DD.xml
      </file-location>
    </archive-descriptor>
    <files>
      <entry-name>
        META-INF/MANIFEST.MF
      </entry-name>
      <file-path>
        \one_example\War_war.mf
      </file-path>
      <entry-name>
        WEB-INF/classes/test1/TestServlet.class
      </entry-name>
      <file-path>
       ./WEB-INF/classes/test1/TestServlet.class
      </file-path>
    </files>
    <files>
     <entry-name>
      index.html
     </entry-name>
     <file-path>
      ./index.html
     </file-path
    </files>
    <directories>
      <dir-path>
        \deploy_example\WEB-INF\classes\test1
      </dir-path>
      <extension-mapping>
        <ext>
          .class
        </ext>
        <mapping>
          WEB-INF/classes/test1
        </mapping>
      </extension-mapping>
      <included>
        YES
      </included>
      <excluded-files>
      </excluded-files>
    </directories>
    <additional-classpath>
      C:\J2EEngine\deploying\deploy_example;
    </additional-classpath>
  </archive>
</j2ee-components-generator>
```

Generate J2EE Components Using Own Java Classes

To generate JAR and WAR files, you can develop your own Java class. To do so, use `com.inqmy.deploy.components.J2EEComponentsGenerator` class file.

J2EE components generator class consists of the following methods:

`public J2EEComponentsGenerator(RandomAccessFile out)` – constructs a `J2EEComponentsGenerator` object with log file. The `out` parameter specifies the random access to the file.

`public J2EEComponentsGenerator(String s)` – constructs a `J2EEComponentsGenerator` object and starts logging significant events. The `s` parameter is a string that specifies the name of the log file.

`public void addNewArchive(String archiveName, int type)` – specifies a new archive with `archiveName` name and type of component to be added to it. The `type` parameter must have values corresponding to the constants defined in `com.inqmy.descriptors.ear.J2EEModule` class – `J2EEModule.EJB`, `J2EEModule.WEB`, `J2EEModule.JAVA` and `J2EEModule.OTHER`. The method adds some of the defined components to a JAR file with the `archiveName` name. The `archiveName` parameter specifies the name of the archive to which the component is added. The `type` parameter specifies the type of component to be added to the archive file.

`public void loadExistingArchive(String filePath) throws J2EEGeneratorException` – loads the archive file with `filePath`, checks the type of archive, and adds it to the `archiveDescriptors` table. The `filePath` parameter specifies the path of the archive file. If an error occurs the method throws a `J2EEGeneratorException`.

`public void addComponent(String archiveName, Descriptor descr)` – adds a component to an existing archive file with the `archiveName` name. This class adds a deployment descriptor to the archive as well. The `archiveName` parameter specifies the name of the archive to which the component is added. The `descr` parameter specifies the descriptor of `com.inqmy.services.deploy.ear.common.Descriptor` type to be added to the archive.

`public void removeArchive(String archiveName)` – removes an archive file from the table with archive deployment descriptors. The `archiveName` parameter specifies the name of the archive to be removed from the table.

`public void removeComponentFromArchive(String archiveName, String componentName)` – removes the component with the `componentName` name from the archive deployment descriptor named `archiveName`. The `archiveName` parameter specifies name of the archive from which the component is removed. The `componentName` parameter specifies name of the component to be removed from the archive.

`public void moveArchiveDescriptor(String fromArchiveName, String toArchiveName, String compName)` – moves a deployment descriptor from one archive to another. The `fromArchiveName` parameter specifies the name of the archive from which the deployment descriptor must be taken. The `toArchiveName` parameter specifies the name of the archive to which the deployment descriptor must be moved. The `compName` parameter specifies the name of the component's deployment descriptor that must be set.

`public Descriptor getDescriptor(String archiveName, String componentName)` – gets a deployment descriptor from a specified archive file. The `archiveName` parameter specifies the name of the archive in which the component must be sought. The class returns the deployment descriptor for the specified component. The `componentName` parameter provides the name of the component's deployment descriptor that must be returned.

`public void setDescriptor(String archiveName, String componentName, Descriptor descr)` – sets deployment descriptor with `componentName` name to archive with `archiveName` name. The `archiveName` parameter specifies the name of

the archive. The componentName parameter specifies the component's deployment descriptor.

public void replaceDescriptor(String jarName, String componentName, String xmlFile) throws J2EEGeneratorException – replaces the deployment descriptor with componentName name in a JAR. The new deployment descriptor is derived from an XML file. The jarName parameter specifies the name of the JAR's descriptor that must be replaced. The componentName parameter specifies the name of the deployment descriptor of application component. The xmlFile parameter specifies the file that contains XML representation of the deployment descriptor. If an error occurs the method throws a J2EEGeneratorException.

public void makeArchive(String archiveName) throws IOException – creates an archive file. The archiveName parameter specifies the name of the new archive. If an error occurs the class throws an IOException.

public void addAdditionalFiles(String archiveName, InfoObject[] infoes) – adds a file to the hashtable of additional files for the archive. This table is used, when a new archive is made, to set additional files. The archiveName parameter specifies the name of archive. The infoes parameter specifies the InofObject arrays that must be set for the additional files.

public void addAdditionalFiles(String archiveName, String[] filePaths, String[] entryNames) – adds files that are specified with two arrays – entry name and file path. The archiveName parameter specifies the name of the archive. The filePaths parameter specifies the array of file paths to the additional file. The entryNames parameter specifies the array of entry names of additional files.

public void addAdditionalFiles(String archiveName, String filePath, String entryName) – adds one *InfoObject* class to the current archive. It contains two parameters – absolute path, and entry name of the additional file. The archiveName parameter specifies the name of the archive. The filePaths parameter specifies the array of file paths to the additional file. The entryNames parameter specifies the array of entry names of additional files.

public void addAdditionalClassPath(String archiveName, String[] paths) – adds a classpath to the archive. It is used for searching some of the classes in classpaths directories. The paths parameter specifies the array of classpaths.

public void setArchiveDescriptor(String archiveName, ArchiveDescriptor archive) – sets an archive deployment descriptor in the hashtable of archives. The archiveName parameter specifies the name of the archive to be added to the table. The archive parameter specifies the content of the archive with archiveName name.

public void setArchiveDescriptor(String archiveName, String xmlFileLocation) throws J2EEGeneratorException – sets archive deployment descriptor in the table of the archive for archiveName and file location of XML representation of ArchiveDescriptor. The archiveName parameter specifies the name of the archive. The xmlFileLocation parameter specifies the location of the XML file. If an error occurs the class throws a J2EEGeneratorException.

public void toLog(String str) – writes a string into the log file. The str parameter specifies the message that must be written in the log file.

public void setLog(RandomAccessFile out) – writes the log file to random access file. The out parameter specifies the file to which the log is printed.

`public void startLog(String s)` – starts logging into the specified log file. The `s` parameter specifies the pathname to the file where the log event must be set.

`public void startLog()` – starts logging into the default log *logfile.txt*.

`public void stopLog()` – closes the log file.

`public void setHasLogFile(boolean type)` – specifies the availability of the log file.

`public void doStuffy()` – deletes information that has been set to the J2EE components generator.

Example:

This example describes the *<SAPj2eeEngine_install_dir>/docs/examples/deploy/comp_generator/CompGeneratorExample.java*. This class uses an EJB and a Servlet to generate an EJB JAR and a WAR. The names and the components are the same as those used in the *Getting Started with Deploy Tool* part of this document (except *TestWar.war*). You can use the source code for your own needs. To do so, perform the following steps:

Step 1

To generate the EJB JAR you must set the `private void ejbGeneration()` method correctly in the *CompGeneratorExample.java*.

Set the name and the full file path of the JAR. The JAR is generated in the *<SAPj2eeEngine_install_dir>/docs/examples/deploy/comp_generator* directory. Its name is – *TestJar.jar*.

Set the Bean home interface class name, remote interface class name and enterprise Bean class name:

```
String beanHomeName = "test1.StatefulOkHomeBean";
String beanRemoteName = "test1.StatefulOkBean";
String ejbName = "test1.TheBeanImpl";
```

Set the Bean name in the JAR to be `SessionBean`.

```
String beanName = "SessionBean";
```

Set the bean type – session or entity:

```
String beanType = "sessionbean";
```

Step 2

Set the `private void webGeneration()` method correctly in the *CompGeneratorExample.java* to generate the WAR file.

Set the name and the full file path of the WAR. The WAR is generated in the *<SAPj2eeEngine_install_dir>/docs/examples/deploy/comp_generator* folder. Its name is *TestWar.war*.

Set the WAR components. The *TestWar.war* archive contains one Servlet and one HTML file.

Set the Servlet name, display name and Servlet class:

```
String testServletName = "TestServlet";
String testServletDisplayName = "TestServlet";
String testServletPackage = "test1.TestServlet";
```

Specify the mapping name of the Servlet in the WAR.

The Servlet is *<SAPj2eeEngine_install_dir>/deploying/deploy _example/WEB-INF/classes/test1/TestServlet.class*. Its mapping must be *WEB-INF/classes/test1/TestServlet.class*:

```
String entryName = "WEB-INF/classes/test1/TestServlet.class";
```

Specify the mapping and the full or the relative path to the HTML file, if your application has some. The WAR contains an *index.html*. The HTML file is *<SAPj2eeEngine_install_dir>/deploying/deploy_example/index.html*. Its mapping name must be *index.html*. The example specifies the entry name and the relative path of the file.

```
filePath = path + "\\index.html";
entryName = "Index.html";
```

Step 3

Specify the application EJB references. The application can have more than one reference; therefore, the `EJBeanReference` is an array. For example:

```
    ejbReference  = new EJBeanReference[1];
    ejbReference[0] = new EJBeanReference(beanType, 0,
beanHomeName, beanRemoteName);
    ejbReference[0].setReferenceLink(beanName);
    webDescriptor.setEjbRefs(ejbReference);
```

Step 4

Set the root directory to the Bean classes' packages: *<SAPj2eeEngine_install_dir>/deploying/deploy_example/* as `String path`.

Step 5

Set the log file name and location. For example:

```
String outputFile = "CompGenerator\\J2EE_Generator_Log.txt";
```

Compiling

To generate J2EE components using the *CompGeneratorExample.java*, the file must be compiled. You must set the following classpaths:

<SAPj2eeEngine_install_dir>/deploying/lib/inqmyxml.jar.

<SAPj2eeEngine_install_dir>/deploying/lib/deploy.jar.

<SAPj2eeEngine_install_dir>/deploying/lib/ejb11.jar.

<SAPj2eeEngine_install_dir>/deploying/lib/servlet.jar

Running

Set the path to the *CompGeneratorExample.class* file to run it: *<SAPj2eeEngine_install_dir>/docs/examples/deploy/comp_generator/CompGeneratorExample.class.*

After the necessary changes are made, start the example using the *<SAPj2eeEngine_install_dir>/docs/examples/deploy/comp_generator* script file.

Note: Because you are using *CompGeneratorExample.class*, three files are generated in the *<SAPj2eeEngine_install_dir>/docs/examples/deploy/comp_generator* folder – *TestWar.war*, *TestJar.jar* and *J2EE_Generator_Log.txt*.

Application Assembling

To deploy an application on SAP J2EE Engine, the application components must be assembled in an EAR file. This file consists of EJB JAR files, client JAR files, and WAR files.

Assembling Using Deploy Tool

The assembling of JARs and WARs in an EAR can be performed using SAP J2EE Engine Deploy Tool. This is described in the *Deploy Tool* section of this document.

Assembling Using Script Files

To pack the JAR and WAR files in an EAR (Enterprise Archive) file you can use the *eargenerator* script file, located in *<SAPj2eeEngine_install_dir>/deploying* directory.

When executing the script file, specify the following arguments on the command line:

- `<XML-file>` – the path to the XML file that the *eargenerator* script file uses to create the EAR file. The structure of this XML file is described below.
- `[LOG-file]` – the file where all log messages are stored. If this argument is not specified, a file named *ear_generator_log.txt* is created by default in the *<SAPj2eeEngine_install_dir>/deploying* directory and the log messages are stored in it.

Execute the script file and its arguments on the command line:

```
eargenerator <XML-file> [LOG-file]
```

There is a built-in validity verification for each tag, so you cannot generate an EAR with incorrect XML file. It is made during the XML parsing and in case of incorrect content an exception is thrown. The procedure terminates and writes the reason for interruption to the log file.

The XML file used by the *eargenerator* script file must have the following data type description:

```
<!DOCTYPE ear-generator [
<!ELEMENT ear-generator (ear)+>
<!ELEMENT ear (project-dir?, pathname, displayname,
(jar|war|client|file)+)>
<!ELEMENT project-dir (#PCDATA)>
<!ELEMENT jar (pathname, entryname, altdd*)>
<!ELEMENT client (pathname, entryname, altdd*)>
<!ELEMENT altdd (pathname, elname)>
<!ATTLIST altdd use-alternative (yes|no) #REQUIRED>
<!ELEMENT file (pathname, entryname)>
<!ELEMENT war (pathname, entryname, contextroot, altdd*)>
<!ELEMENT elname (#PCDATA)>
<!ELEMENT pathname (#PCDATA)>
<!ELEMENT entryname (#PCDATA)>
<!ELEMENT displayname (#PCDATA)>
<!ELEMENT contextroot (#PCDATA)>
]>
```

All fields are explained below:

Note: The DOCTYPE field must be specified properly for the XML file to be parsed.

```
<!--
Ear-generator element is the root element.
This element is obligatory and the name has to be ear-generator.
-->
<!ELEMENT ear-generator (ear)+>

<!--
Ear element describes the EAR file to be generated by the
eargenerator script file. The jar|war|client|file element
specifies the type of the file added to the EAR.
Used in: ear-generator
-->
<!ELEMENT ear (project-dir?, pathname, displayname,
(jar|war|client|file)+)>

<!--
Project-dir element specifies the project directory where project
files are located. This is a main element for the application
components described with relative file path. If this element is
not specified the location of the application components have to
be specified by their full file path.
Used in: ear
-->
<!ELEMENT project-dir (#PCDATA)>

<!--
Jar element describes a JAR file in the EAR file.
Used in: ear
-->
<!ELEMENT jar (pathname, entryname, altdd*)>

<!--
Client element describes the client archive file in the EAR.
Used in: ear
-->
<!ELEMENT client (pathname, entryname, altdd*)>

<!--
The altdd element specifies an alternative application component
deployment descriptor to replace the original application
component deployment descriptor. Altdd element specifies the
component level deployment descriptor – JAR descriptor, WAR
descriptor and client JAR descriptor. There is no limit on the
number of the altdd elements. Use the element to make changes at
deployment descriptor of the element.
Used in: war, client, jar
-->
<!ELEMENT altdd (pathname, elname)>

<!--
Altdd attribute specifies whether the alternative deployment
descriptor, to be used. Possible values are yes or no. If you set
“no” don’t specify pathname and elname for altdd element.
-->
<!ATTLIST altdd use-alternative (yes|no) #REQUIRED>

<!--
File element specifies the additional file(s) to be added to the
EAR.
Used in: ear
```

```
-->
<!ELEMENT file (pathname, entryname)>

<!--
War element describes the WAR files in the EAR.
Used in: ear
-->
<!ELEMENT war (pathname, entryname, contextroot, altdd*)>

<!--
The elname element specifies the name of the component for which
an alternative deployment descriptor to be used. For example name
of a Bean.
Used in: altdd
-->
<!ELEMENT elname (#PCDATA)>

<!--
The pathname element specifies the full or relative file path.
Used in: ear, jar, client, altdd, file, war
-->
<!ELEMENT pathname (#PCDATA)>

<!--
The entryname element specifies the name the file uses in the EAR.
The name the file uses in the local file system may differ from
its name in the EAR file.
Used in: jar, client, file, war
-->
<!ELEMENT entryname (#PCDATA)>

<!--
The displayname element specifies the name the EAR file uses in
the J2EE Engine namespace, by which the J2EE Engine looks for it.
Used in: ear
-->
<!ELEMENT displayname (#PCDATA)>

<!--
The contextroot element specifies the WAR context root, by which
the HTTP Service accesses the WAR.
Used in: war
-->
<!ELEMENT contextroot (#PCDATA)>
```

Example:

To build an EAR file from example components in *<SAPj2eeEngine_install_dir>/deploying/deploy_example* use the script file *<SAPj2eeEngine_install_dir>/deploying/eargenerator*.

Execute the script file and its arguments on the command line:

```
eargenerator <XML-file> [LOG-file]
```

Where:

- `<XML-file>` – the path to the XML file *<SAPj2eeEngine_install_dir>/deploying/deploy_example/eargen.xml*

- `[LOG-file]` – the file where log messages are stored. If this argument is not specified, a file named *ear_generator_log.txt* is created in the *<SAPj2eeEngine_install_dir>/deploying* directory by default.

An Example for XML file used by the *eargenerator* script file follows:

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE ear-generator [
<!ELEMENT ear-generator (ear)+>
<!ELEMENT ear (project-dir?, pathname, displayname,
(jar|war|client|file)+)>
<!ELEMENT project-dir (#PCDATA)>
<!ELEMENT jar (pathname, entryname, altdd*)>
<!ELEMENT client (pathname, entryname, altdd*)>
<!ELEMENT altdd (pathname, elname)>
<!ATTLIST altdd use-alternative (yes|no) #REQUIRED>
<!ELEMENT file (pathname, entryname)>
<!ELEMENT war (pathname, entryname, contextroot, altdd*)>
<!ELEMENT elname (#PCDATA)>
<!ELEMENT pathname (#PCDATA)>
<!ELEMENT entryname (#PCDATA)>
<!ELEMENT displayname (#PCDATA)>
<!ELEMENT contextroot (#PCDATA)>
]>

  <ear-generator>
    <ear>
      <project-dir>
        C:\J2EEngine\deploying
      </project-dir>
      <pathname>
        \one_example\Assemble.ear
      </pathname>
      <displayname>
        za_na_ryka
      </displayname>
      <jar>
        <pathname>
          \one_example\jar.jar
        </pathname>
        <entryname>
          jar.jar
        </entryname>
        <altdd use-alternative="yes">
          <pathname>
            \one_example\jar_jarAltDD.xml
          </pathname>
          <elname>
            sessionbean
          </elname>
        </altdd>
        <altdd use-alternative="yes">
          <pathname>
            \one_example\jar_jarAltDD.xml
          </pathname>
          <elname>
            jar.jar
          </elname>
        </altdd>
      </jar>
      <client>
        <pathname>
          \one_example\Client.jar
        </pathname>
        <entryname>
```

```
        Client.jar
      </entryname>
    </client>
    <war>
      <pathname>
        \one_example\War.war
      </pathname>
      <entryname>
        War.war
      </entryname>
      <contextroot>
        War
      </contextroot>
      <altdd use-alternative="yes">
        <pathname>
          \one_example\War_war_AltDD.xml
        </pathname>
        <elname>
          War.war
        </elname>
      </altdd>
      <altdd use-alternative="yes">
        <pathname>
          \one_example\War_war_AltDD.xml
        </pathname>
        <elname>
          TestServlet
        </elname>
      </altdd>
    </war>
  </ear>
</ear-generator>
```

Assembling Using Own Java Classes

The EAR generator `com.inqmy.deploy.generator.XMLEarGenerator` interface is used to pack the JAR and WAR files in an EAR (Enterprise Archive) file.

EAR generator interface consists of the following methods:

`public void addJarArchive(String path, String entry) throws FileNotFoundException;` – adds a JAR archive to the EAR. The file path to the existing JAR must be specified, as well as the entry name of the JAR in the EAR. The `path` parameter specifies the path to the existing JAR. The `entry` parameter specifies the entry name of the JAR in the EAR. This is the name the JAR has in the EAR.

`public void addClientArchive(String path, String entry) throws FileNotFoundException;` – adds a client JAR archive to the EAR. The file path to the existing JAR must be specified, as well as the entry name of the JAR in the EAR. The `path` parameter specifies the path to the existing JAR. The `entry` parameter specifies the entry name of the JAR in the EAR. This is the name the JAR has in the EAR. If an error occurs a `FileNotFoundException` is thrown.

`public void addOtherFile(String path, String entry) throws FileNotFoundException;` – adds a file that is not a JAR or WAR archive to the EAR. The path of the existing file and the entry name of the file in the EAR must be specified. The `path` parameter specifies the path to the existing file. The `entry` parameter specifies the entry name of the file in the EAR. This is the name the file has in the EAR. If an error occurs a `FileNotFoundException` is thrown.

`public void addWarArchive(String path, String entry, String root) throws FileNotFoundException;` – adds a WAR archive to the EAR. The file path, entry name, and context root of the existing file must be specified. The `path` parameter specifies the path to the file. The `entry` parameter specifies the entry name of the file in the EAR. This is the name the file has in the EAR. The `root` parameter specifies the WAR Context root. If an error occurs a `FileNotFoundException` is thrown.

`public EJBeanJAR getEJBeanJARDescriptor(String jarName) throws EARGeneratorException;` – returns the deployment descriptor of a JAR file from the EAR archive. The `jarName` parameter specifies the entry name of the JAR file. The class returns the deployment descriptor of the JAR. If an error occurs an `EARGeneratorException` is thrown.

`public DeploymentDescriptor getEJBBeanDescriptor(String jarName, String beanName) throws EARGeneratorException;` – returns the deployment descriptor of a specified Bean into the specified JAR. The `jarName` parameter specifies the entry name of the JAR that contains the Bean. The `beanName` parameter specifies the name of the Bean whose deployment descriptor must be retrieved. If an error occurs an `EARGeneratorException` is thrown.

`public ApplicationClientDescriptor getAppClientDescriptor(String jarname) throws EARGeneratorException;` – returns the deployment descriptor of a JAR file from the EAR archive. The `jarName` parameter specifies the entry name of the JAR file. The class returns the JAR deployment descriptor. If an error occurs an `EARGeneratorException` is thrown.

`public WebDeploymentDescriptor getWarDescriptor(String warname) throws EARGeneratorException;` – returns the deployment descriptor of a WAR file from the EAR archive. The `warname` parameter specifies the WAR file entry name.

`public void setEJBeanJarDescriptor(EJBeanJar ejbean, String jarname);` – replaces the enterprise bean deployment descriptor of a JAR from the EAR. The `ejbean` parameter specifies the new EJB-JAR deployment descriptor. The `jarname` parameter specifies the JAR entry name.

`public void setEJBeanDeploymentDescriptor(DeploymentDescriptor dd, String jarname);` – replaces a deployment descriptor from the specified JAR file. The Bean name is taken from the specified deployment descriptor. The `dd` parameter specifies the new deployment descriptor that must replace an old one with the same name. The names are compared by using the `getName()` method from the deployment descriptor interface. The `JARname` parameter specifies the JAR name.

`public void setAppClientDescriptor(EJBeanJar ejbean, String jarname);` – replaces the application client deployment descriptor of a JAR from the EAR. The `ejbean` parameter specifies the new EJB-JAR deployment descriptor. The `jarname` parameter specifies the JAR entry name.

`public void setWarDescriptor(WebDeploymentDescriptor war, String warName);` – replaces the deployment descriptor of a WAR from the EAR. The `ejbean` parameter specifies the new web application deployment descriptor that must be set to the WAR. The `warName` parameter specifies the entry name of the WAR in the EAR.

`public void replaceBeanDescriptor(String jarname, String beanname, String xmlfile) throws EARGeneratorException;` – replaces the Bean deployment descriptor from the JAR file by reading the descriptor from the specified

XML file. The XML file is a standard EJB-JAR deployment descriptor that can contain several Bean descriptors. Only the deployment descriptor specified by `beanname` is taken. The `jarname` parameter specifies the entry name of the JAR in the EAR. The `beanname` parameter specifies the name of the Bean whose deployment descriptor must be replaced. The `xmlfile` parameter specifies the XML file that contains the deployment descriptors.

`public void startEar(String displayname);` – starts the creation of an EAR archive with the specified display name. The `displayname` parameter specifies the display name for the EAR archive.

`public boolean makeEar(String earFileName);` – creates the EAR after some archives have been added. The `earFileName` parameter specifies the file name (including path) of the EAR that must be created.

`public void setDisplayName(String value);` – sets the display name of the EAR. The `value` parameter specifies the new display name.

Note: In the display name, do not use the following special symbols: slash (/), double backslash (\\), double quotation (“), colon (:), asterisk (*), question mark (?), backslash (\), greater than sign (>), less than sign (<) and vertical slash (|). They will be replaced with underscore (_). Do not end the string with period (.).

`public String getDisplayName();` – returns the display name. The `string` contains the display name.

Example:

This example uses a WAR and an EJB JAR to generate an EAR file. The components names are the same as those used in the *Getting Started with Deploy Tool* part of this document (except *TestWar.war*). You can use the source for your own needs. The source code is *<SAPj2eeEngine_install_dir>/docs/examples/deploy/ear_maker/EarMakerExample.java* file. It uses WAR and JAR files generated with the example source described in the *Generate J2EE Components Using Own Java Classes* part of this document. If the *CompGeneratorExample.class* is not used, the specifications of WAR and JAR files in the following example source must be changed.

Step 1

Set the parent directory. This is the folder positioned at the beginning of the relative paths of the components:

<SAPj2eeEngine_install_dir>/docs/examples/deploy/ear_maker

You must set the parent directory properly.

Step 2

Set the WAR and EJB JAR files to be assembled in the EAR file. In the example, they are *TestJar.jar* and *TestWar.war* respectively:

```
String[] ejbJarFiles = new String[]
{"CompGenerator\\TestJar.jar"};
String[] warFiles = new String[] {"CompGenerator\\TestWar.war"};
```

Step 3

Set the EAR file output name, display name and location:

```
String outputName = parentDir + "TestEAR.ear";
String earName = "ExampleEAR";
```

Step 4

Set the log file, to be used to reflect the process of EAR making. Here it is *EarLog.txt*:

```
String logFile = parentDir + "EarLog.txt";
```

Compiling

To make the EAR file using *EarMakerExamp.java*, the file must be compiled. Set the following classpaths:

<SAPj2eeEngine_install_dir>/deploying/lib/deploy.jar

<SAPj2eeEngine_install_dir>/deploying/lib/inqmyxml.jar

Running

Set the file path to the *EarMakerExample.class* file:

<SAPj2eeEngine_install_dir>/docs/examples/deploy/ear_maker/

After the necessary changes are made, start the example using the *<SAPj2eeEngine_install_dir>/docs/examples/deploy/EarMakerExample* script file.

Note: Because you are using the *EarMakerExample* script file, two files are generated in the *<SAPj2eeEngine_install_dir>/docs/examples/deploy/ear_maker* folder – *TestEAR.ear* and *earLog.txt*.

Deployment

After completing the assembling process, you must specify the user security roles, user mappings, context roots, and so on. You can use the Deploy Tool, or you can write a specific XML file manually, use Shell commands, or your own Java classes.

Hint: To make an application default starting for the J2EE Engine, the context root of the application must be set as "/." This overrides the default-starting page in the HTTP root directory.

Deployment Using Deploy Tool

To deploy applications with SAP J2EE Engine Deploy Tool, refer to the *Deploy Tool* section of this document.

Deployment Using Script Files

To deploy your application you can use the *deploymanager* script file, located in the *<SAPj2eeEngine_install_dir>/deploying* directory.

Note: A line in each script file in *<SAP_j2eeEngine_install_dir>/deploying* folder indicates that the SAP J2EE Engine parser is used. The command is `-Dserver.parser.inqmy=true`.
The JAR file is *<SAP_j2eeEngine_install_dir>/deploying/lib/inqmyxml.jar*. If you want to use a different parser, provide a different classpath to a JAR file created in accordance with the JAXP specification, and remove this line from the starting script.

When executing the script file you have to specify the following arguments on the command line:

- `<XML-file>` – the path to the XML file, that the *deploymanager* script file will use to deploy the application. In this XML file specify the security roles, user mapping (mapping security roles to real users), context root, and so on . The structure of the XML file is described bellow.
- `<Log-file>` – the file where all log messages will be stored. If this argument is not specified, a file named *deploy_manager_log.txt* will be created by default in the *<SAPj2eeEngine_install_dir>/deploying* directory and the log messages will be stored in it.

Execute the script file and its arguments on the command line:

```
deploymanager <XML-file> [<Log-file>]
```

When the *deploymanager* script file is executed, validity verification for each tag in the XML file is performed and in case of incorrect content an exception is thrown. The procedure terminates and writes to the log file the reason for interruption.

The XML file used by the *deploymanager* script file must have the following data type description:

```
<!DOCTYPE DeployMgmInf [
<!ELEMENT DeployMgmInf (project-dir?, xml-ear-path, supports,
storage-properties?, user-role-mapping, version-id?, additional-
```

```
info?,login-info, log-file?, library-info?, deployment-
properties?)>
<!ELEMENT project-dir (#PCDATA)>
<!ELEMENT xml-ear-path (#PCDATA)>
<!ATTLIST xml-ear-path action-type (deploy|update) #REQUIRED>
<!ELEMENT supports (support+)>
<!ELEMENT support (#PCDATA)>
<!ELEMENT type (#PCDATA)>
<!ELEMENT storage-properties (EntityEJBStorages)>
<!ELEMENT EntityEJBStorages (EntityEJB)*>
<!ELEMENT EntityEJB (Storage)>
<!ATTLIST EntityEJB Name CDATA #REQUIRED>
<!ELEMENT Storage (RDBStorageProps)>
<!ATTLIST Storage StorageName CDATA #REQUIRED ClassName CDATA
#REQUIRED>
<!ELEMENT RDBStorageProps (FieldColumnMap*, rdbFinderDescriptor*)>
<!ATTLIST RDBStorageProps PoolName CDATA #REQUIRED TableName CDATA
#REQUIRED>
<!ELEMENT FieldColumnMap (#PCDATA)>
<!ATTLIST FieldColumnMap Field CDATA #REQUIRED Table CDATA
#REQUIRED Column CDATA #REQUIRED Key CDATA #REQUIRED>
<!ELEMENT rdbFinderDescriptor (MethodArgument*)>
<!ATTLIST rdbFinderDescriptor MethodName CDATA #REQUIRED Criteria
CDATA #REQUIRED >
<!ELEMENT MethodArgument (#PCDATA)>
<!ELEMENT user-role-mapping (role-name+)>
<!ELEMENT role-name (#PCDATA|mapping)*>
<!ELEMENT mapping (name,type)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT login-info (host, port, transport-protocol*, user-name,
user-password)>
<!ELEMENT host (#PCDATA)>
<!ELEMENT port (#PCDATA)>
<!ELEMENT transport-protocol (#PCDATA)>
<!ELEMENT user-name (#PCDATA)>
<!ELEMENT user-password (#PCDATA)>
<!ELEMENT version-id (#PCDATA)>
<!ELEMENT additional-info (archives)>
<!ELEMENT archives (ejb-jar*, war*, client-jar*)>
<!ELEMENT ejb-jar (archive-name, ejbean+)>
<!ELEMENT ejbean (bean-name, ejb-ref*, res-ref*, entry*)>
<!ELEMENT war (archive-name, ejb-ref*, res-ref*, entry*, context-
param*)>
<!ELEMENT client-jar (archive-name, ejb-ref*, res-ref*, entry*)>
<!ELEMENT archive-name (#PCDATA)>
<!ELEMENT bean-name (#PCDATA)>
<!ELEMENT entry (entry-name, value)>
<!ELEMENT entry-name (#PCDATA)>
<!ELEMENT value (#PCDATA)>
<!ELEMENT res-ref (ref-name, ref-link, res-user, res-password)>
<!ELEMENT ref-name (#PCDATA)>
<!ELEMENT ref-link (#PCDATA)>
<!ELEMENT res-user (#PCDATA)>
<!ELEMENT res-password (#PCDATA)>
<!ELEMENT ejb-ref (ref-name, ref-link)>
<!ELEMENT context-param (param-name, param-value, description)>
<!ELEMENT param-name (#PCDATA)>
<!ELEMENT param-value (#PCDATA)>
<!ELEMENT description (#PCDATA)>
<!ELEMENT log-file (#PCDATA)>
<!ELEMENT library-info (referenced-library*, deployable-library* ,
add-refs*)>
<!ELEMENT referenced-library (#PCDATA)>
<!ELEMENT deployable-library (lib-name, lib-jar*)>
<!ELEMENT lib-name (#PCDATA)>
<!ELEMENT lib-jar (#PCDATA)>
```

```
<!ATTLIST lib-jar server-map-name CDATA #REQUIRED>
<!ELEMENT add-refs (fromLoader , toLoader+)>
<!ELEMENT fromLoader (#PCDATA)>
<!ELEMENT toLoader (#PCDATA)>
<!ELEMENT deployment-properties (property*)>
<!ELEMENT property (#PCDATA)>
]>
```

All fields are explained below:

Note: The DOCTYPE field must be specified properly for the XML file to be parsed correctly.

```
<!--
DeployMgmInf element is the root element.
This element is obligatory and the name has to be DeployMgmInf.
 -->
<!ELEMENT DeployMgmInf (project-dir?, xml-ear-path, supports,
storage-properties?, user-role-mapping, version-id?, additional-
info?,login-info, log-file?, library-info?, deployment-
properties?)>

<!--
Project-dir element specifies the project directory, where project
files are located. This is a main element for the application
components described with relative file path. If this element is
not specified the location of the application components have to
be specified by their full file path.
Used in: DeployMgmInf.
-->
<!ELEMENT project-dir (#PCDATA)>

<!--
The xml-ear-path element specifies the file path to the
application EAR file used for deployment. The file path can be
full or relative path. If the project-dir element is not specified
use full file path. This element is obligatory and critical for
execution of the deploymanager script file.
Used in: DeployMgmInf.
-->
<!ELEMENT xml-ear-path (#PCDATA)>

<!--
Action-type attribute specifies the deployment action. Possible
values are deploy and update. Use deploy to deploys a new
application. Use update to update an already existing application.
During update procedure, if the application contains new
components, they are added to the deployed application. This
attribute is not obligatory.
Used in: xml-ear-path.
-->
<!ATTLIST xml-ear-path action-type (deploy|update) #REQUIRED>

<!--
Supports element specifies list of remote support types to be
generated.
Used in: DeployMgmInf.
-->
<!ELEMENT supports (support+)>

<!--
Support element specifies the support type.
Currently, the available support is p4. If no support is set, the
application will be deployed with p4 support. There is an RMI/IIOP
support for deployment of EJB applications, but to use this
```

```
functionality, make sure the IIOP service is running both on
dispatcher and server nodes.
Used in: supports.
-->
<!ELEMENT support (#PCDATA)>

<!--
Storage-properties element describes storage properties of entity
EJB with container persistent management. It is optional for the
DeployMgmInf XML file. Specify it only in case that the
application contains entity EJB.
Used in: DemployMgmInf.
-->
<!ELEMENT storage-properties (EntityEJBStorages)>

<!--
EntityEJBStorages element contains descriptions of storage
properties of entity EJB.
Used in: storage-properties.
-->
<!ELEMENT EntityEJBStorages (EntityEJB)*>

<!--
EntityEJB element defines entity EJB that has storage properties.
Used in: EntityEJBStorages.
-->
<!ELEMENT EntityEJB (Storage)>

<!--
Name attribute specifies the Bean name of an entity Bean.
The name of the Bean is unique for the application and is critical
for the correct deployment of the application.
Used in: EntityEJB.
-->
<!ATTLIST EntityEJB Name CDATA #REQUIRED>

<!--
Storage element specifies all RDBStorageProps properties.
Used in: EntityEJB.
-->
<!ELEMENT Storage (RDBStorageProps)>

<!--
StorageName and ClassName attribute specifies the storage type and
the class that implements it. Currently, the only available
storage type is RDB Storage and the available ClassName is
com.inqmy.services.ejb.deploy.descriptors.dbstorage.RDBMSStoragePr
operties.
Used in: Storage.
-->
<!ATTLIST Storage StorageName CDATA #REQUIRED ClassName CDATA
#REQUIRED>

<!--
RDBStorageProps element specifies RDBStorageProps by pool and
table name.
Used in: Storage.
-->
<!ELEMENT RDBStorageProps (FieldColumnMap*, rdbFinderDescriptor*)>

<!--
PoolName and TableName attributes specify the pool name and table
name of the used database. To deploy the application properly be
sure that Pool and Table with specified names exist at the SAP
J2EE Engine.
Used in: RDBStorageProps.
```

```
-->
<!ATTLIST RDBStorageProps PoolName CDATA #REQUIRED TableName CDATA
#REQUIRED>

<!--
FieldColumnMap element defines the correspondence between the
table fields of the database and the EJB's fields.
Used in: RDBStoragePrpos.
-->
<!ELEMENT FieldColumnMap (#PCDATA)>

<!--
Field, Table, Column and Key attributes describes the field in the
entity EJB. For example they explain where to map the bean. Key
specifies if this field is a primary key or a regular table
column. The possible values are "1" and "0" respectively.
Used in: FieldColumnMap.
-->
<!ATTLIST FieldColumnMap Field CDATA #REQUIRED Table CDATA
#REQUIRED Column CDATA #REQUIRED Key CDATA #REQUIRED>

<!--
RdbFinderDescriptor element describes the type of the find method.
Used in: RDBStoragePrpos.
-->
<!ELEMENT rdbFinderDescriptor (MethodArgument*)>

<!--
MethodName and Criteria attributes describe the find method and
search criteria. MethodName specifies the name of the method. A
Criteria is the SQL statement for the find method and describes
the WHERE clause of the SELECT statement. It is not obligatory to
set Criteria for findByPrimaryKey method. If the Criteria is
empty, the whole table fields are selected. If the Criteria begins
with select, the whole Criteria is read as a SQL statement. At the
places where the value of the parameters have to be set, use $
(Dollar sign) in front of the value. Number the parameters equal
to or bigger than one.
Used in: rdbFinderDescriptor.
-->
<!ATTLIST rdbFinderDescriptor MethodName CDATA #REQUIRED Criteria
CDATA #REQUIRED>

<!--
MethodArgument element defines the argument type of a method to be
recognized by all find methods. For example – int.
Used in: rdbFinderDescriptor.
-->
<!ELEMENT MethodArgument (#PCDATA)>
```

Note: The elements storage-properties, EntityEJBStorages, EntityEJB, Storage, RDBStorageProps, FieldColumnMap, rdbFinderDescriptor, MethodArgument are required only for entity Beans with container-managed persistence.

```
<!--
User-role-mapping describes list of security roles.
Used in: DeployMgmInf.
-->
<!ELEMENT user-role-mapping (role-name+)>

<!--
Role-name element defines the role name, that is stored in the
deployment descriptor's XML, to be mapped to the real users.
Used in: user-role-mapping.
```

```
-->
<!ELEMENT role-name (mapping*)>

<!--
Mapping element defines mappings to users or groups (or both),
defined in the J2EE Engine User Management System.
Used in: user-role-mapping.
-->
<!ELEMENT mapping (name, type)>

<!--
Name element specifies the name of a user or group.
Used in: mapping.
-->
<!ELEMENT name (#PCDATA)>

<!--
Type element defines the type of the security user. Possible
values are: user or group.
Used in: mapping.
-->
<!ELEMENT type (#PCDATA)>

<!--
Login-info element sets information needed to connect to the J2EE
Engine and to lookup the J2EE Engine deploy services. This element
is critical for the deployment procedure.
Used in: DeployMgmInf.
-->
<!ELEMENT login-info (host, port, transport-protocol*, user-name,
user-password)>

<!--
Host element specifies the dispatcher host name to connect to SAP
J2EE Engine.
Used in: login-info.
-->
<!ELEMENT host (#PCDATA)>

<!--
Port element specifies the port that for connection to the
cluster.
Used in: login-info.
-->
<!ELEMENT port (#PCDATA)>

<!--
Transport-protocol element specifies the protocol name by which
the connection is realized.
DeployManager uses SSL and Http Tunneling protocols. If you set
SSL be sure that the SSL service already is running at SAP J2EE
Engine. Possible values are SSL and httptunneling.
Used in: login-info.
-->
<!ELEMENT transport-protocol (#PCDATA)>

<!--
User-name element specifies the name of a registered user with
administrator right.
Used in: login-info.
-->
<!ELEMENT user-name (#PCDATA)>

<!--
User-password specifies the password of a registered user.
Used in: login-info.
```

```
-->
<!ELEMENT user-password (#PCDATA)>

<!--
Version-id element specifies the version number of SAP J2EE Engine
that deploys the application.
Used in: DeployMgmInf.
-->
<!ELEMENT version-id (#PCDATA)>

<!--
Additonal-info element describes the additional information about
the application. Changes in EJB reference, resource reference and
environment entry parameters in deployment descriptors can be made
during deploy. Only the values of already existing parameters can
be changed – no additional parameters can be added.
Used in: DeployMgmInf.
-->
<!ELEMENT additional-info (archives)>

<!--
Archives element specifies the type of the archive file in the
EAR.
Used in: additional-info.
-->
<!ELEMENT archives (ejb-jar*, war*, client-jar*)>

<!--
Ejb-jar element describes the EJB JAR by its archive name and EJB.
Used in: archives.
-->
<!ELEMENT ejb-jar (archive-name, ejbean+)>

<!--
Ejbean element specifies the EJB by its name, reference, resource
reference and entry attribute.
Used in: ejb-jar.
-->
<!ELEMENT ejbean (bean-name, ejb-ref*, res-ref*, entry*)>

<!--
War element describes the WAR file by its name, EJB references,
resource reference and environment entries.
Used in: archives.
-->
<!ELEMENT war (archive-name, ejb-ref*, res-ref*, entry*, context-
param*)>

<!--
Client-jar element specifies the client JAR by its name, EJB
reference, resource reference and entry attributes.
Used in: archives.
-->
<!ELEMENT client-jar (archive-name, ejb-ref*, res-ref*, entry*)>

<!--
Archive-name element specifies the name of the archive file.
Used in: ejb-name, war, client-jar.
-->
<!ELEMENT archive-name (#PCDATA)>

<!--
Bean-name element specifies the EJB name.
Used in: ejbean.
-->
<!ELEMENT bean-name (#PCDATA)>
```

```
<!--
Entry element – describes an environment entry by name and value.
The type of the entry cannot be specified on this stage.
Used in: ejbean, war, client-jar.
-->
<!ELEMENT entry (entry-name, value)>

<!--
Entry-name element specifies the name of the environment entry.
Used in: entry.
-->
<!ELEMENT entry-name (#PCDATA)>

<!--
Value element specifies the value of the environment entry.
Used in: entry.
-->
<!ELEMENT value (#PCDATA)>

<!--
Res-ref element describes a resource reference of a file by its
resource name, reference link, username (the name of the user that
has access to the code), and password.
Used in: ejbean, war, client-jar.
-->
<!ELEMENT res-ref (ref-name, ref-link, res-user, res-password)>

<!--
Ref-name element specifies the name of the environment entry used
in the code.
Used in: res-ref, ejb-ref.
-->
<!ELEMENT ref-name (#PCDATA)>

<!--
Ref-link element specifies the name of an EJB in the J2EE
application package, i.e. the name of the EJB in the JNDI.
Used in: res-ref, ejb-ref.
-->
<!ELEMENT ref-link (#PCDATA)>

<!--
Res-user element specifies the name of a user that has access to
the code.
Used in: res-ref.
-->
<!ELEMENT res-user (#PCDATA)>

<!--
Res-password element specifies the user password.
Used in: res-ref.
-->
<!ELEMENT res-password (#PCDATA)>

<!--
Ejb-ref element describes the EJB reference by reference name and
reference link.
Used in: ejbean.
-->
<!ELEMENT ejb-ref (ref-name, ref-link)>

<!--
Context-param element specifies the WAR context root, under which
the WAR will be accessible from the HTTP Service.
Used in: war.
```

```
-->
<!ELEMENT context-param (param-name, param-value, description)>

<!--
Param-name element specifies the name of the context parameter.
Used in: context-param.
-->
<!ELEMENT param-name (#PCDATA)>

<!--
Param-value element specifies the value of the context parameter.
Used in: context-param.
-->
<!ELEMENT param-value (#PCDATA)>

<!--
Description element describes a text about a context parameter.
Used in: context-param.
-->
<!ELEMENT description (#PCDATA)>

<!--
Log-file element specifies the file all log messages to be stored
in.
Used in: DeployMgmInf.
-->
<!ELEMENT log-file (#PCDATA)>

<!--
Library-info element specifies the library that is necessary for
the application.
Used in: DeployMgmInf.
-->
<!ELEMENT library-info (referenced-library*, deployable-library*,
add-refs*)>

<!--
Referenced-library element specifies the library, already deployed
on the J2EE Engine that will be used as a reference to the current
application.
Used in: library-info.
-->
<!ELEMENT referenced-library (#PCDATA)>

<!--
Deployable-library element specifies the library that will be
deployed along with the application each time the application is
deployed.
Used in: library-info.
-->
<!ELEMENT deployable-library (lib-name, lib-jar*)>

<!--
Lib-name element specifies the name of the library to be deployed.
Used in: deployable-library.
-->
<!ELEMENT lib-name (#PCDATA)>

<!--
Lib-jar element specifies the path to the library archive.
Used in: deployable-library.
-->
<!ELEMENT lib-jar (#PCDATA)>

<!--
Lib-jar attribute specifies mapping name for library jar files.
```

```
The goal is these libraries to be deployed to the J2EE Engine and
to be used from other applications later too. You can modify the
way the libraries to be deployed arranging them in subdirectories
to avoid JAR files collision.
Used in: lib-jar.
-->
<!ATTLIST lib-jar server-map-name CDATA #REQUIRED>

<!--
Add-refs element supplements a reference from an application,
library or service to another application, library or service.
This element is not obligatory. For example, use this element if
you deploy application that uses a library, but this library uses
another library, service or application.
Used in: library-info.
-->
<!ELEMENT add-refs (fromLoader ,toLoader+)>

<!--
FromLoader element specifies the library, application or service
that uses the reference. The possible values are: application:
<application_name>, library: <library_name>, service:
<service_name>.
This element is obligatory on add-refs element.
Used in: add-refs.
-->
<!ELEMENT fromLoader (#PCDATA)>

<!--
ToLoader element specifies the library, application or service
that is used from the reference.The possible values are:
application: <application_name>, library: <library_name>, service:
<service_name>.
This element is obligatory on add-refs element
Used in: add-refs.
-->
<!ELEMENT toLoader (#PCDATA)>

<!--
Deployment-properties element specifies the properties, like J2EE™
Specification, root lookup, container type.
Used in: DeployMgmInf.
-->
<!ELEMENT deployment-properties (property*)>

<!--
Property element must be specified as “key=value.” Use one of the
following: container_type = B or container_type = A; root_lookup =
false or root_lookup = true; specification = J2EE_1_2 or
specification = J2EE_1_3;
Used in: deployment-properties.
-->
<!ELEMENT property (#PCDATA)>
```

`[Log-file]` – specifies the full path to the log file. If not specified, a *log.txt* file is created in the current folder.

Example:

To deploy an EAR file from the example components in *<SAPj2eeEngine_install_dir>/deploying/deploy_example* use the script file *<SAPj2eeEngine_install_dir>/deploying/deploymanager*.

Execute the script file and its arguments on the command line:

```
deploymanager <XML-file> [LOG-file]
```

Where:

- `<XML-file>` – the path to the XML file *<SAPj2eeEngine_install_dir>/deploying/deploy_example/deploy.xml*
- `[LOG-file]` – the file where log messages are stored. If this argument is not specified, a file named *deploy_manager_log.txt* is created in the *<SAPj2eeEngine_install_dir>/deploying* directory by default.

Note: To execute the example properly, start the SAP J2EE Engine.

An example of XML file used by the *deploymanager* script file follows:

```
<DeployMgmInf>
    <project-dir>
      C:\SAP J2EE Engine\deploying
    </project-dir>
    <xml-ear-path>
      \example\Yeah.ear
    </xml-ear-path>
    <supports>
      <support>
        P4
      </support>
    </supports>
<storage-properties>
 <EntityEJBStorages>
  <EntityEJB Name="entitybean">
      <Storage
ClassName="com.inqmy.descriptors.ejb.dbstorage.RDBMSStorageProperties" StorageName="RDB Storage">
        <RDBStorageProps PoolName="BEANPOOL" TableName="AUTHORS">
         <rdbFinderDescriptor Criteria="" MethodName="findAll">
         </rdbFinderDescriptor>
          <rdbFinderDescriptor Criteria="LASTNAME = $1"
MethodName="findByLastName">
            <MethodArgument>
              java.lang.String
            </MethodArgument>
          </rdbFinderDescriptor>
        </RDBStorageProps>
      </Storage>
    </EntityEJB>
 </EntityEJBStorages>
</storage-properties>
<user-role-mapping>
 <role-name>
   Hello
   <mapping>
    <name>
    Administrator
    </name>
    <type>
     user
    </type>
   </mapping>
   <mapping>
    <name>
     guests
    </name>
```

```
    <type>
     group
    </type>
   </mapping>
 </role-name>
</user-role-mapping>
    <version-id>
    </version-id>
    <additional-info>
      <archives>
        <ejb-jar>
          <archive-name>
            TestJar.jar
          </archive-name>
          <ejbean>
            <bean-name>
              sessionbean
            </bean-name>
          </ejbean>
        </ejb-jar>
        <war>
          <archive-name>
            WebArchive.war
          </archive-name>
          <ejb-ref>
            <ref-name>
              sessionbean
            </ref-name>
            <ref-link>
              sessionbean
            </ref-link>
          </ejb-ref>
        </war>
      </archives>
    </additional-info>
    <login-info>
      <host>
        localhost
      </host>
      <port>
        3011
      </port>
      <transport-protocol>
        httptunneling
      </transport-protocol>
      <user-name>
        Administrator
      </user-name>
      <user-password>
      </user-password>
    </login-info>
    <library-info>
      <deployable-library>
        <lib-name>
          library:IAIKS
        </lib-name>
        <lib-jar>
          C:\ SAP J2EE Engine \additional-lib\iaik_jsse.jar
        </lib-jar>
        <lib-jar>
          C:\ SAP J2EE Engine \additional-lib\iaik_ssl.jar
        </lib-jar>
      </deployable-library>
      <referenced-library>
        library:IAIKS
      </referenced-library>
```

```
        </library-info>
        <deployment-properties>
          <property>
            container_type = B
          </property>
        </deployment-properties>
    </DeployMgmInf>
```

If an exception occurs, the deployment process terminates. The reason for the interruption is recorded in the log file.

Deployment Using Shell Commands

To deploy applications using SAP J2EE Engine Shell console, refer to the *Shell Commands Reference* section in the Administration Manual.

Deployment Using Visual Administrator

SAP J2EE Engine Visual Administrator can be used for modifying the deployed applications. The usage of this tool is described in the Administration Manual.

Deployment Using Own Java Classes

DeployManager's `com.inqmy.deploy.manager.DeployManagerImpl` class is used to deploy, undeploy and update applications and libraries on the J2EE Engine. It provides methods to start and stop deployed applications, and to reference them to external libraries (for example JAR files). The deployment process requires login information, the file path to the application to be deployed, and some additional information stored in property files (usually XML files). To provide this additional information:

- Use a property file. You must set the properties manually.
- Use a set of methods.

DeployManager interface provides the following methods:

`public final static byte EJB_JAR = 1;` – the byte value represents components of type *ejb.jar*.

`public final static byte WEB_WAR = 2;` – the byte value represents components of type *web.war*.

`public final static byte CLIENT_JAR = 3;` – the byte value represents components of type *client.jar*.

`public void setEar(String earPath);` – sets the path to the EAR file to be deployed. The file path can be absolute or relative to the project directory. The `earPath` parameter is the file path to the EAR file.

`public void setSupport(String[] support);` – sets supported communication protocols for the application. If this property is not specified, the EAR EJB is deployed with the default remote protocol – `P4`. If the specified support is not available in the J2EE Engine, the value is ignored. The `support` parameter lists the supported protocols.

`public void setStorageProperties(PersistentStorageProperties storageProperties, String jarName, String beanName) throws IllegalArgumentException, IncorrectJ2EEException;` – sets some additional data about persistent storage of an entity EJB. Storage properties must be specified for each entity EJB with container-managed persistence, otherwise it is not deployed. The `storageProperties` parameter represents the Bean persistent storage. The `JarName` parameter is the name of the JAR file where the Bean is located. The `BeanName` parameter represents the name of the Bean to which these properties belong. If no Bean exists with specified name or it is not an entity Bean an `IllegalArgumentException` is thrown. If the EAR deployment descriptor is `null` or incomplete an `IncorrectJ2EEException` is thrown.

`public void setUserRoleMapping(String userRole, String userName, boolean type);` – sets the mapping between roles and users or groups. For a defined security role, its JAR and WAR must have the same user mapping. The `userRole` parameter names the security role. The `userName` parameter names the user or group to be mapped to this role. The `type` parameter represents the user type. It is `true` if the type is `user` and `false` if the type is `group`.

`public void setLoginInfo(LoginInfo info);` – sets login information necessary to establish a connection to the J2EE Engine. The `info` parameter represents the information necessary for login to the J2EE Engine.

`public void setLog(String logFilePath);` – sets the path to the log file in which the (whole) information about the deployment process is saved. The `logFilePath` parameter is the name of the log file.

`public void setDeployerVersion(String ver);` – sets the version of the property file used for deployment. DeployManager uses it for compliance between different versions. This class sets the string specifying the version ID.

`public String getDeployerVersion();` – returns the version of the property file used for deployment. It returns a string specifying the version ID.

`public String getEar();` – returns the path to the EAR file to be deployed.

`public String[] getSupport();` – returns a list of transport protocols available for the application.

`public PersistentStorageProperties getStorageProperties(String jarName, String beanName) throws IllegalArgumentException, IncorrectJ2EEException;` – returns the storage persistence properties of the specified entity container-managed EJB. The `jarName` parameter specifies the name of the JAR file where this Bean is located. The `beanName` parameter specifies the name of the EJB. If bean with this name does not exist or the bean is not entity an `IllegalArgumentException` is thrown. An `IncorrectJ2EEException` is thrown if the EAR deployment descriptor is null or incomplete.

`public Hashtable getUserRoleMappings();` – returns the mapping between security roles and user information. Information about the user name and the user type to every security role is mapped here. The type can be `user` or `group`. Keys of the table are the names of the security roles. Their values are of `UserRoleMappings` type and define the user name and type.

`public LoginInfo getLoginInfo();` – returns the login information necessary to connect to the J2EE Engine. These are the current values of the `LoginInfo` as read from properties, or set by the corresponding `set` method.

`public String getLog();` – returns the path to the log file. This file contains error and debug information for the processes that `DeployManager` performs.

`public void setProjectDirectory(String dir);` – sets the name of the project directory. All paths to resource files, EARs, and so on, for the `DeployManager` can be specified as relevant from this directory. The `dir` parameter specifies the project directory.

`public String getProjectDirectory();` – loads the name of the project directory. All paths to resource files, EARs, and so on, for the `DeployManager` can be specified as relevant from this directory, except JAR files for application libraries. The `dir` parameter specifies the project directory.

`public String[] getComponents(byte type) throws IllegalArgumentException, IncorrectJ2EEException;` – loads the names of EAR's components with the specified type. This type can be `EJB_JAR`, `WEB_WAR` or `CLIENT_JAR`. The `type` parameter specifies the type of the components. This class returns an array of component names from the specified type. If the type is not one of the defined constants an `IllegalArgumentException` is thrown. If the EAR deployment descriptor is `null` or incomplete an `IncorrectJ2EEException` is thrown.

`public void loadProperties(File propertiesFile) throws DeployManagerException;` – loads the information, necessary for the deployment process, from the XML file. The `propertiesFile` parameter specifies this file. If a problem occurs during loading the property file a `DeployManagerException` is thrown.

`public void storeProperties(File propertiesFile) throws DeployManagerException;` – stores information into a property file necessary for the deployment process. Typically, this is an XML file. It stores the property's changes made from the methods for one session before deployment or update. The `propertiesFile` parameter specifies a file in which the deploy information is stored. If a problem during storing the information occurs a `DeployManagerException` is thrown.

`public EnvironmentEntry[] getEnvironmentEntries(String componentName, byte type) throws IllegalArgumentException, IncorrectJ2EEException;` – loads the environment entries for a component from a property file. It is used to receive detailed information about environment entries, such as name, description, type and value. The `componentName` parameter specifies the name of a component whose environment entries are required. The `type` parameter specifies the type of the component – `EJB_JAR`, `WEB_WAR` or `CLIENT_JAR`. This class returns a list of environment entries for the specified component. If a component with this name does not exist an `IllegalArgumentException` is thrown.

`public void setEnvironmentEntries(String componentName, byte type, EnvironmentEntry[] newEntries) throws IllegalArgumentException, WarningException, IncorrectJ2EEException;` – sets the environment entries for a component. Each environment entry value can be changed. It cannot be removed or deleted. The `componentName` parameter specifies the name of the component to which environment entries are set. The `type` parameter specifies the type of the component – `EJB_JAR`, `WEB_WAR` or `CLIENT_JAR`. The `newEntries` parameter specifies the list of environment entries to be set to the specified component. If a component with this name does not exist an `IllegalArgumentException` exception is thrown. If the specified environment entry does not exist a `WarningException` exception is thrown. If the EAR deployment descriptor is `null` or incomplete an `IncorrectJ2EEException` is thrown.

`public ResourceReference[] getResourceReferences(String componentName, byte type) throws IllegalArgumentException, IncorrectJ2EEException;` – loads the resource references for a component from a property file. It is used to receive detailed information about resource references, such as name and type of the reference and resource link, and description. The `componentName` parameter specifies the name of the component for which resource references are required. The `type` parameter specifies the type of the component – `EJB_JAR`, `WEB_WAR` or `CLIENT_JAR`. This class returns a list of resource references for the specified component. If a component with this name does not exist an `IllegalArgumentException` is thrown.

`public void setResourceReferences(String componentName, byte type, ResourceReference[] newResources) throws IllegalArgumentException, WarningException, IncorrectJ2EEException;` – sets the resource references for a component. Each resource link can be changed. It cannot be removed or deleted. The `componentName` parameter specifies the name of the component for which resource references are set. The `type` parameter specifies the type of the component – `EJB_JAR`, `WEB_WAR` or `CLIENT_JAR`. The `newRecources` parameter displays a list of resource references to be set to the specified component. If a component with this name does not exist an `IllegalArgumentException` exception is thrown. If the specified resource reference does not exist a `WarningException` exception is thrown. If the EAR deployment descriptor is `null` or incomplete an `IncorrectJ2EEException` exception is thrown.

`public EJBeanReference[] getEJBeanReferences(String componentName, byte type) throws IllegalArgumentException, IncorrectJ2EEException;` – loads the EJB references for a component from a property file. This class gets detailed information about EJB references, such as name, type and description of the reference, names of home and remote interface of the referenced Bean, and reference link. The `componentName` parameter specifies the name of the component whose EJB references are required. The `type` parameter specifies the type of the component – `EJB_JAR`, `WEB_WAR` or `CLIENT_JAR`. This class returns a list of EJB references for the specified component. If a component with this name does not exist an `IllegalArgumentException` is thrown. If the EAR deployment descriptor is `null` or incomplete an `IncorrectJ2EEException` exception is thrown.

`public void setEJBeanReferences(String componentName, byte type, EJBeanReference[] newReferences) throws IllegalArgumentException, WarningException, IncorrectJ2EEException;` – sets the EJB references for a component. Each link can be changed, but it cannot be removed or deleted. The `componentName` parameter specifies the name of the component for which EJB references are set. The `type` parameter specifies the type of the component – `EJB_JAR`, `WEB_WAR` or `CLIENT_JAR`. The `newReferences` parameter specifies the list of EJB references to be set to the specified component. If a component with this name does not exist an `IllegalArgumentException` is thrown. If the specified EJB reference does not exist a `WarningException` is thrown. If the EAR deployment descriptor is `null` or incomplete an `IncorrectJ2EEException` is thrown.

`public ContextParam[] getContextParams(String componentName) throws IllegalArgumentException, IncorrectJ2EEException;` – loads the context parameters for a Web component from a property file. It is used to retrieve detailed information about Servlet context parameters, such as name, value and description. The `componentName` parameter specifies the name of the component whose EJB references are required. This class returns a list of context parameters for the specified component. If a component with this name does not exist an `IllegalArgumentException` is thrown. If the EAR deployment descriptor is null or incomplete an `IncorrectJ2EEException` is thrown.

`public void registerLibraryReference(String[] toLibraries);` – registers references to libraries to be used by this application. The names of the libraries can be an existing library or the one to be deployed with the application. The `toLibraries` parameter lists the libraries referenced by this application.

`public void registerApplicationLibrary(String libName, String[] jars, String[] mappings);` – registers library for an application. Each library is a `name-value` pair. The `name` is a string identifier of the library name and the `value` contains a string array of paths to all files belonging to the library. These libraries must be deployed on the J2EE Engine, and also be used from other applications. You can modify the way the libraries are deployed by arranging them in subdirectories to avoid JAR files collision. The `libName` parameter specifies the name of the library. The `jars` parameter specifies a list of libraries to be registered for this application. The `jars` string must be a full file path, not a relative one. The `mappings` string specifies a list of libraries where jar files can be deployed on the J2EE Engine. The paths must be relative. If the `mappings` string is missing or the array is `null` the JAR files are copied to the default location *<SAPj2eeEngine_install_dir>/additional-lib/*.

`public void deployLanguageLib (String app, String[] jars)` – deploys language libraries to ensure multilingual support of the applications. They can be deployed during the application deployment process, or later. The `app` parameter specifies the name of the application where libraries are to be deployed. The `jars` parameter specifies the array of JAR names of libraries to be deployed.

`public void setContextParams(String componentName, ContextParam[] newParams) throws IllegalArgumentException, WarningException, IncorrectJ2EEException;` – sets the context parameters for a Web component. Each context parameter can be changed, but it cannot be removed or deleted. The `componentName` parameter specifies the name of the component for which context parameters are set. The `newParams` parameter specifies a list of context parameters to be set to the specified component. If a component with this name does not exist an `IllegalArgumentException` is thrown. If the specified context parameter does not exist a `WarningException` is thrown. If the EAR deployment descriptor is `null` or incomplete an `IncorrectJ2EEException` is thrown.

`public boolean isContainerAType();` – checks if this entity Bean in the application is deployed in a container of type `A` or type `B`. A container of type `A` uses one instance for each primary key through all transactions that are currently involved with this primary key. If the application container is of type `A` this class returns `true`; and if the application container is of type `B` returns `false`.

`public void registerLibraryReference(String[] toLibraries);` – registers references to libraries used by the application. The names of the libraries can be of an existing library or of a library to be deployed with the application. The `toLibraries` parameter specifies a list of libraries referenced by this application.

`public void registerApplicationLibrary(Hashtable toLibraries);` – registers libraries for this application. Each library has a `name-value` pair specification. The `name` is a string identifier of the library name and the `value` contains a string array of paths to all files belonging to this library. These libraries are deployed to the J2EE Engine and can be used from other applications. The `toLibraries` parameter describes a list of libraries to be registered for this application.

`public void deploy() throws DeployManagerException, java.rmi.RemoteException;` – deploys an application. The deployment process needs information specified using the methods of DeployManager interface – login information, file path to the application to be deployed, property file, containing

deploy information, and so on. If problems occur during the deployment process a `DeployManagerException` is thrown. If a remote problems occur during the deployment process a `RemoteException` is thrown.

`public void update() throws DeployManagerException, java.rmi.RemoteException;` – updates an application. A real update must implement changes to the current version of an already deployed application with the different files only. The important point is to take into account all settings that are persistent for the application and the ones that are to be overridden in the current version. If a problem occurs during update process a `DeployManagerException` is thrown. If a remote problem occurs during the update process a `RemoteException` is thrown.

`public void undeploy(String applicationName) throws DeployManagerException, RemoteException;` – undeploys the specified application. It removes all the components that belong to this application from the naming and from the corresponding file structure. After undeployment, the application is not accessible. The `applicationName` parameter specifies the name of the application to be undeployed. If a problem occurs during the undeployment process a `DeployManagerException` exception is thrown. If a remote problem occurs during the undeployment process a `RemoteException` is thrown.

`public void startApplication(String applicationName) throws DeployManagerException, RemoteException;` – starts an application that is deployed but is in a “Stop” status. The `applicationName` parameter specifies the name of the application to be started. If a problem occurs during the start process a `DeployManagerException` is thrown. If a remote problem occurs during the start process or the specified application has not been deployed a `RemoteException` is thrown.

`public void stopApplication(String applicationName) throws DeployManagerException, RemoteException;` – stops a deployed client-side application. The application resides on the J2EE Engine, but is not accessible until it is started. The `applicationName` parameter specifies the name of the application to be stopped. If errors occur during the stop process a `DeployManagerException` is thrown. If a remote problem occurs during the stop process a `RemoteException` is thrown.

`public void deployLibrary(String libName, String[] jars) throws RemoteException;` – deploys a library. Each library has name-value pair specification. The `name` is a string identifier of the library name and the `value` defines a string array of paths to all files belonging to this library. The `libName` parameter specifies the name of this library. The `jars` parameter lists all JAR files that belong to this library. Strings representing the paths to the files specify the JAR files. If a problem occurs during deployment process a `RemoteException` is thrown.

`public void deployLibrary(String libName, String[] jars, String[] mappings) throws RemoteException;` – deploys a library. Each library is a `name-value` pair. The `name` is a string identifier of the library name and the `value` defines a string array of paths to all files belonging to the library. These libraries must be deployed on the J2EE Engine, and also be used from other applications. You can modify the way the libraries are deployed by arranging them in subdirectories to avoid JAR files collision. The `libName` parameter specifies the name of the library. The `jars` parameter specifies a list of libraries to be registered for this application. The `jars` string must be a full file path, not a relative one. The `mappings` string specifies a list of libraries where jar files are deployed on the J2EE Engine. The paths must be relative. If the `mappings` string is missing or the array is `null` the JAR files are copied

to the default location *<SAPj2eeEngine_install_dir>/additional-lib/*. If problems during deployment process occur a `RemoteException` is thrown.

`public void removeLibrary(String libName) throws RemoteException;` – removes an existing library with the specified name from the J2EE Engine. The `libName` parameter specifies the name of this library. If a problem occurs during the remove process a `RemoteException` is thrown.

`public void updateLibrary(String libName, String[] jars) throws RemoteException;` – updates the library with the specified name, replacing its old contents with the new specified JAR files, or adding new ones. The `libName` parameter specifies the name of the library to be updated. The `jars` parameter specifies the list of the JAR files that represents the new library content. Strings that represent the paths to the files specify the JAR files. If a problem occurs during the update process the `RemoteException` is thrown.

`public void updateLibrary(String libName, String[] jars, String[] mappings) throws RemoteException;` – updates the library with the specified name, replacing its old contents with the new specified JAR files, or adding new ones. The `libName` parameter specifies the name of the library for update. The `jars` parameter specifies a list of jar files representing the new library contents. The strings representing the paths to the files specify the JAR files. If a problem during the update process occurs a `RemoteException` is thrown.

`public void makeReferences(String fromApplication, String[] toLibraries) throws RemoteException;` – makes references from an application to the specified libraries. The `fromApplication` parameter specifies the name of the application. The `toLibraries` parameter specifies the list of library names to be referenced by this application. If a problem occurs during reference making a `RemoteException` is thrown.

`public File getCurrentStatusXML(String applicationName) throws DeployManagerException, RemoteException;` – loads the `XML` file that stores information for the application properties. The `applicationName` parameter specifies the name of the application whose property file is required. The class returns a property file that stores the deploy information. If a problem occurs during XML generation process a `DeployManagerException` is thrown. If a remote problem occurs during XML generation process a `RemoteException` is thrown.

`public ExportInfo[] getCurrentStatus(String applicationName) throws DeployManagerException, RemoteException` – gets information about the current status of the specified application. An `ExportInfo` object represents each of the application components. It holds information about components' properties. The `applicationName` parameter specifies the name of the application whose status is required. The method returns an array of `ExportInfos`. If a problem during the XML generation process occurs a `DeployManagerException` is thrown. If a problem during getting application status occurs a `RemoteException` is thrown.

`public String getApplicationStatus(String appName) throws RemoteException` – returns the status of an application. The `appName` is the name of the application whose status is returned. If a problem occurs during getting the application status a `RemoteException` is thrown.

`public File getClientJar(String appName, String outputDir) throws RemoteException` – gets the application client jar and saves it in a given output directory. The `appName` is the name of the application. The `outputDir` is the output directory where the client JAR is saved.

`public Properties getDeployProperties ();` – gets additional properties from DeployManager. These properties describe information about container type, root look up parameter, and specification.

`public void removeReferences(String fromApplication, String[] toLibraries) throws RemoteException` – removes references from an application to specified libraries. The `fromApplication` parameter specifies the name of the application. The `toLibraries` parameter specifies the list of libraries whose references must be removed. If a problem during the remove process occurs a `RemoteException` is thrown.

`public void setDeployProperties(Properties deplProps);` – sets additional properties to DeployManager. These properties describe information about the container type, root look up parameter and specification.

Example:

This example uses an EAR file to deploy an application on SAP J2EE Engine. You must run SAP J2EE Engine to execute the example. The source code is: *<SAPj2eeEngine_install_dir>/docs/examples/deploy/deploy_manager/DeployManagerExample.java*. The names of the components are the same as those used in the *Getting Started with Deploy Tool* part of this document. You can use the source file for your own needs. To do so, perform the following steps. It uses an EAR file generated with the example source described in *Assembling Using Own Java Classes* part of this document. If the *EarMakerExample.class* file has not been used, the specifications of the EAR file in the following example source must be changed.

Step 1

Set the support protocol for the application, which in our case is P4:

```
deployManager.setSupport(new String[]{"P4"});
```

Step 2

Set the login information to connect to SAP J2EE Engine. This information includes remote host, remote port, user name and password:

```
String host         = "localhost";
String userName     = "Administrator";
String userPassword = "";
int port            = 3011;
```

You must set this information in *DeployManagerExample.java*.

Step 3

Set the EAR file to be deployed. Use a full file path. If the file and the *DeployManagerExample* script file are in a common directory, you can use the relative file path:

<SAPj2eeEngine_install_dir>/docs/examples/deploy/ear_maker/TestEAR.ear

You must set this in *DeployManagerExample.java*.

Step 4

Set the log file, to be used to reflect the deployment process:

<SAPj2eeEngine_install_dir>/docs/examples/deploy/deploy_manager/DeployMangerLog.txt

Compiling

To deploy the EAR file using the *DeployManagerExample.java*, the file must be compiled. You must set the following classpaths:

<SAPj2eeEngine_install_dir>/deploying/lib/deploy.jar

<SAPj2eeEngine_install_dir>/deploying/lib/inqmyxml.jar

<SAPj2eeEngine_install_dir>/deploying/lib/ejb11.jar

<SAPj2eeEngine_install_dir>/deploying/lib/servlet.jar

<SAPj2eeEngine_install_dir>/deploying/lib/iq-lib.jar

Running

You must set the file path to the *DeployManagerExample.class* file: *<SAPj2eeEngine_install_dir>/docs/examples/deploy/deploy_manager/.*

Note: Because you are using the *DeployManagerExample* script file, a file is generated in the *<SAPj2eeEngine_install_dir>/docs/examples/deploy/deploy_manager/* directory – *DeployManagerLog.txt.*

Chapter 2

Deploy Tool

- Overview
- Deploy Tool Components
- Generate J2EE Components
- Application Assembling
- Deployment

Overview

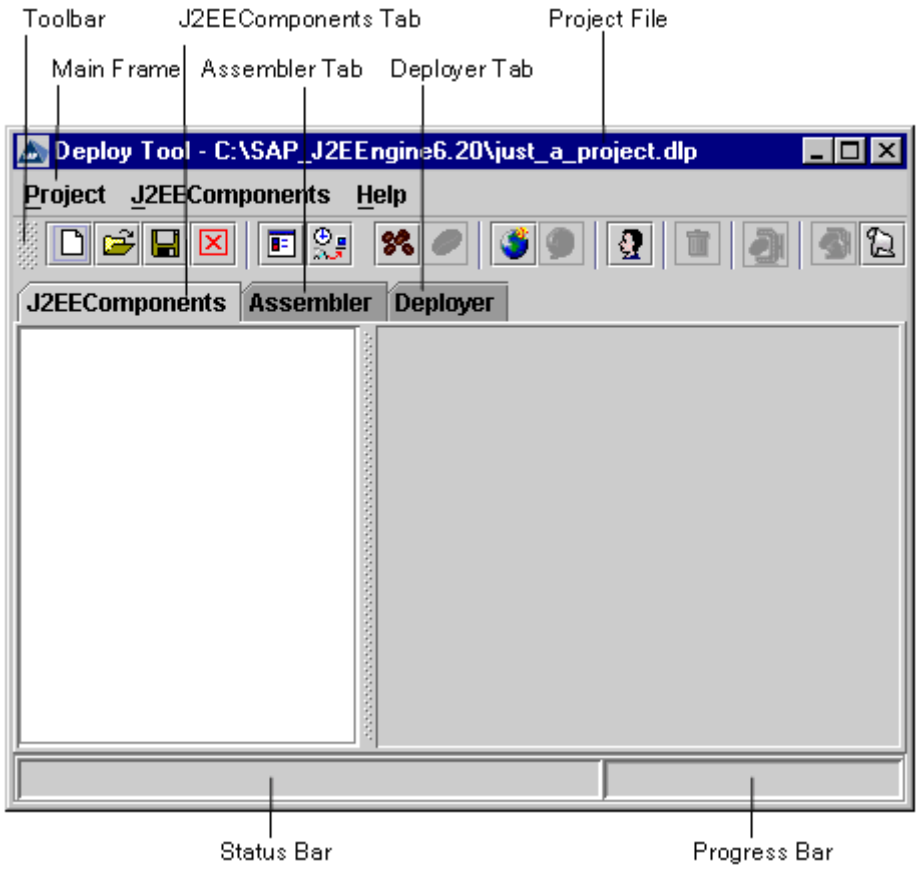
SAP J2EE Engine Deploy Tool is a graphical user interface that assembles archive components and deploys them on SAP J2EE Engine. This tool can be used to create J2EE components out of existing class files and deployment descriptors, assemble J2EE components in an application EAR, and deploy an EAR on the specified SAP J2EE Engine cluster elements.

Deploy Tool is distributed as a part of SAP J2EE Engine, in a component group named “deploying.” To use this tool, you must install it along with SAP J2EE Engine. For more information, refer to the *Installation Manual* document.

To start the Deploy Tool, execute the *<SAPj2eeEngine_install_dir>/deploying/DeployTool* script file, or run it from the program folder *SAP J2EE Engine→Tools→Deploy Tool*.

Deploy Tool Components

Deploy Tool has the following main interface:



Deploy Tool Interface

Deploy Tool consists of three tabs corresponding to the roles in the deployment process – “J2EEComponents,” “Assembler,” and “Deployer.” It also has a status bar, a progress bar, a toolbar and a main frame.

The “J2EEComponents” tab is used to make archive files out of J2EE components. It creates EJB JARs, Web archives (WARs), and ApplicationClient JARs. Information about this tag is stored in an XML file named *Assembler.xml*.

The “Assembler” tab combines JAR and WAR files in an application. At this stage, the deployment descriptors from the original JAR files can be modified, if necessary, and the EAR application file is generated. The information about the JAR and WAR files is stored in a project xml file named *assembler_xml.xml* that is in project directory. For each JAR or WAR file, an alternative deployment descriptor is created. These files store the information necessary to visualize the properties of the “Assembler” tab.

At the deployment stage, performed using the “Deployer” tab, the storage properties and the mappings of users to security roles must be specified. The information about the deployed EAR, the deployment destination, the storage properties, and the assignments of roles to users are also stored in an XML file named *Deployer.xml*.

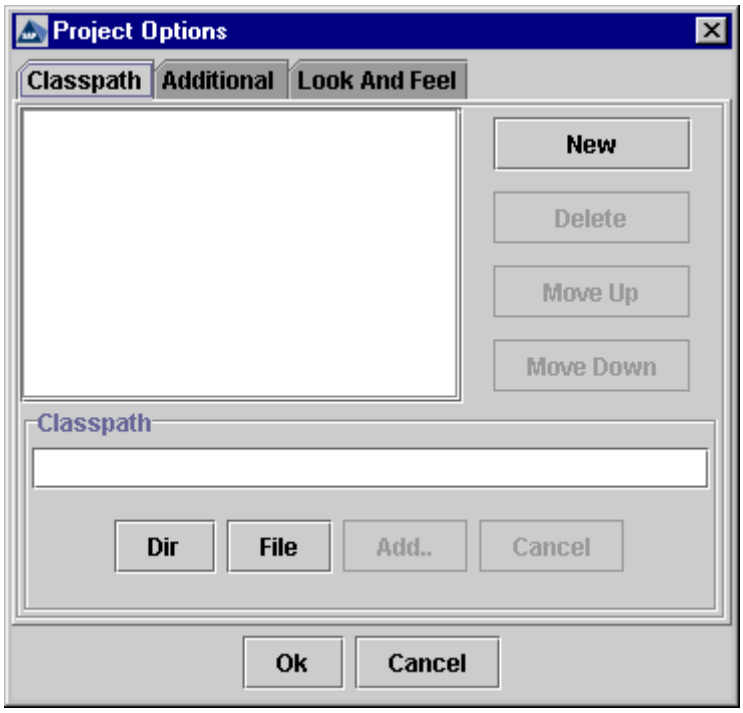
The status bar displays announcements that specify the current condition, the finished task or the process in action.

The progress bar is an active bar that specifies the quantity of accomplished tasks.

Main Frame

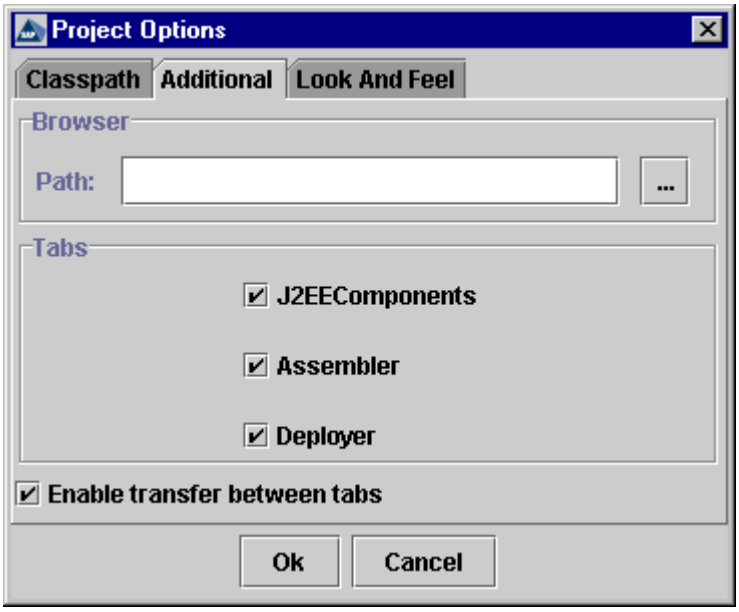
The three tabs – “J2EEComponents,” “Assembler” and “Deployer,” – are in one frame that displays different menu and toolbar options for each tab, according to the tasks performed in it. The “Main” frame contains commands for general management of the project. There is a *Project* menu, in which the following commands are available:

- *New Project* or on the toolbar – creates a new project file with a *dlp* extension. A project folder is created in the same directory and with the same name as the project file. All project files are saved in that folder.
- *Open Project* or on the toolbar – opens an existing *dlp* project file.
- *Save* or on the toolbar – saves the current project.
- *Save as...* – saves the current project under a different name. The content of the current project folder is copied into the new project folder.
- *Close Project* or on the toolbar – closes the active project
- *Options...* or on the toolbar – shows the project options and enables you to edit the project settings. Three tabs are available in the dialog box that appears
  - “Classpath” tab – enables you to set the project classpath. The usage of this tab is crucial for all J2EE components in the project application. More than one classpath can be set. There is a connection between the Java class and the position of the classpath in the classpath list. The classpath for a Java class is searched from the list of available classpaths in order of appearance. The “Move Up” and “Move Down” options can be used to rearrange the position of the available classpaths.



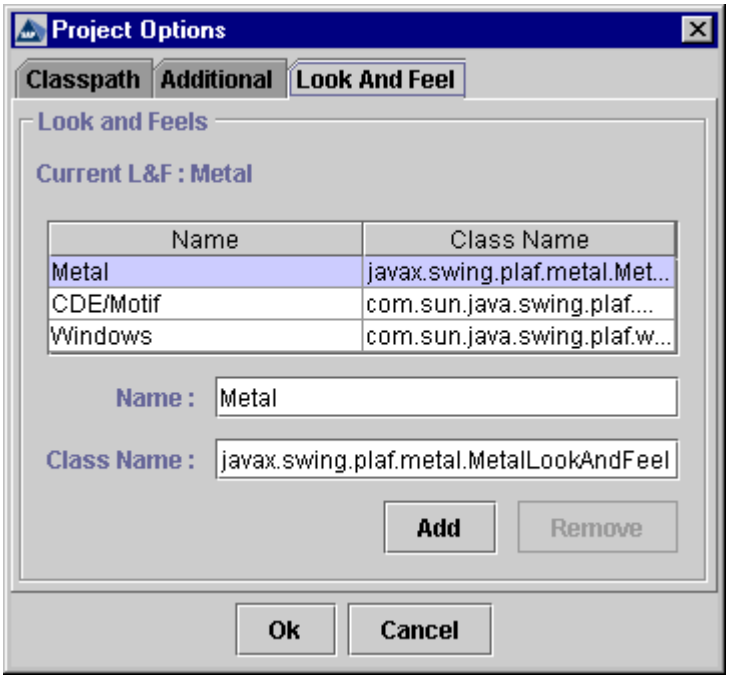
The “Classpath” Tab

- “Additional” tab – shows three checkboxes for each tab of the Deploy Tool. You can choose which tabs are displayed in the Deploy Tool “Main” frame – “J2EEComponents,” “Assembler,” and “Deployer.” At least one tab must be selected. If you deselect the check box for a tab, and this tab contains project information, this information is lost when you save the project. Transfer between tabs can be enabled or disabled. The “Path” field specifies the path to your Internet browser to be used by the Help option.



The “Additional” Tab

- Look And Feel – displays the installed “Look and Feels” and the one in use. Selecting one of the available modes changes the Deploy Tool “Look and Feel.” There are three modes available – “Metal,” “CDE/Motif” and “Windows.” A new “Look And Feel” can be added by specifying its name and class that extends the standard `javax.swing.LookAndFeel`. The default mode is “Metal.”



The “Look And Feel” Tab

- Make All or on the toolbar – undertakes three operations. First, it makes all archives in “J2EEComponents” tab; second, it makes an EAR file in the “Assembler” tab; and finally it deploys the EAR using “Deployer” tab. If you modify the components of an assembled application with a generated EAR file this command is useful. This command generates all JARs, WARs, or EARs once again.
- Project history revision – lists the last five project files that has been opened with Deploy Tool
- Exit – closes Deploy Tool

Keyboard Shortcuts

You can access tasks you perform frequently by using shortcut keys – one or more keys you press simultaneously on the keyboard to complete a task.

There are two types of shortcuts – direct and indirect.

Direct shortcuts provide direct access to the desired command.

Indirect shortcuts provide access to desired commands by means of the menu. If more than one shortcut must be performed to access a command, they are separated by a comma. Care is needed when using shortcuts.

For example, you can open a new project using the Ctrl+N direct shortcut or the Alt+P, Alt+N indirect shortcuts.

Deploy Tool operations and their keyboard shortcuts follow.

Main Frame

Command	Indirect Shortcuts	Direct Shortcut
New Project	Alt+P, Alt+N	Ctrl+N
Open Project	Alt+P, Alt+O	Ctrl+O
Save	Alt+P, Alt+S	Ctrl+S
Save As	Alt+P, Alt+A	
Close Project	Alt+P, Alt+C	
Options...	Alt+P, Alt+T	
Make All	Alt+P, Alt+M	
Project History Revision	Alt+P, Alt+1÷5	
Exit	Alt+P, Alt+X	Ctrl+X

Help

Command	Indirect Shortcuts	Direct Shortcut
Help	Alt+H, Alt+E	F1
About	Alt+H, Alt+U	

J2EEComponents

Command	Indirect Shortcuts	Direct Shortcut
Add EJB Group	Alt+J, Alt+G	
Add EJB	Alt+J, Alt+E	
Add Web	Alt+J, Alt+W	
Add Servlet / JSP	Alt+J, Alt+S	
Add Client	Alt+J, Alt+C	
Remove	Alt+J, Alt+R	Ctrl+R
Make Archive	Alt+J, Alt+A	Ctrl+H
Make All Archives	Alt+J, Alt+M	Ctrl+M
View Log File	Alt+J, Alt+V	

Assemble

Command	Indirect Shortcuts	Direct Shortcut
Add Archive	Alt+A, Alt+D	Ctrl+D
Remove	Alt+A, Alt+R	Ctrl+R
Refresh	Alt+A, Alt+F	
Make Ear	Alt+A, Alt+M	Ctrl+M
View Log File	Alt+A, Alt+V	

Deploy

Command	Indirect Shortcuts	Direct Shortcut
Load Ear	Alt+D, Alt+E, Alt+L	Ctrl+L
Unload Ear	Alt+D, Alt+E, Alt+U	Ctrl+U
Refresh	Alt+D, Alt+E, Alt+R	
Connect	Alt+D, Alt+C	
Deploy Ear	Alt+D, Alt+M, Alt+E	Ctrl+E
Update	Alt+D, Alt+M, Alt+U	Ctrl+P
Undeploy	Alt+D, Alt+M, Alt+N	
Properties	Alt+D, Alt+P	
Libraries	Alt+D, Alt+L	
View Log File	Alt+D, Alt+V	

Generate J2EE Components

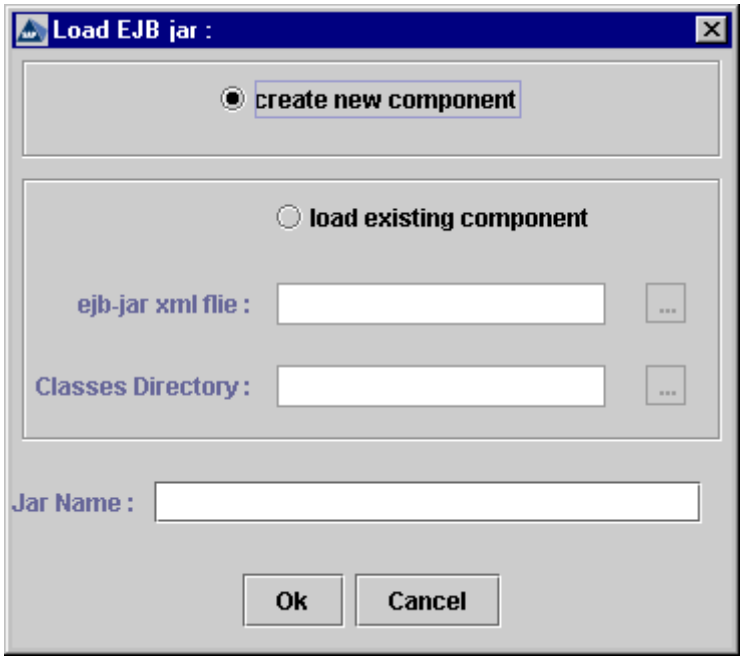
To deploy an application, you must have all the necessary components needed by the client application. These components can be EJB classes, JSPs, Servlets, and all additional files that you need in your project. Your components must be packed in the corresponding archive files, according to the J2EE™ Specification – that is, you must create or provide the JAR and WAR files. The "J2EEComponents" tab is used to create archive files out of J2EE components. The "J2EEComponents" tab makes EJB JARs, Web archives (WARs), and ApplicationClient JARs according to the J2EE™ Specification.

The following commands are available in the *J2EEComponents* menu.

Add EJB Group

The *Add EJB Group* command or  on the toolbar – adds a new or an existing JAR to the project. Two options are available in the dialog box that appears:

- "Create new component" – creates a new JAR file. The "JAR Name" field must be specified. After adding the JAR to the project, at least one EJB must be added to it.
- "Load existing component" – adds a created EJBGroup containing EJBs to the current project. The "EJB-JAR file," Java "Classes Directory" and "JAR Name" fields must be filled in. You must specify the Classes Directory. If the specified Java classes directory is not correct, the EJB Group is loaded but an error message appears. In this case, the classpath must be modified from the *Project→Options* menu "Classpath" tab. This feature is useful when you have many EJBs organized in an EJB Group that must be added to the project.

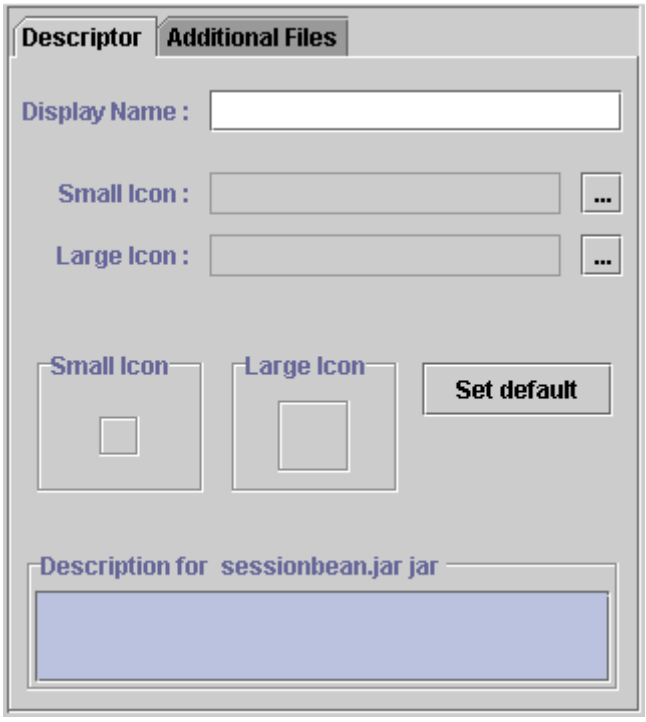


The "Load EJB jar" Window

If the component is added the status bar displays a note. The basic EJBGroup properties can be viewed and edited from the two tabs in the right-hand pane – “Descriptor” and “Additional Files.”

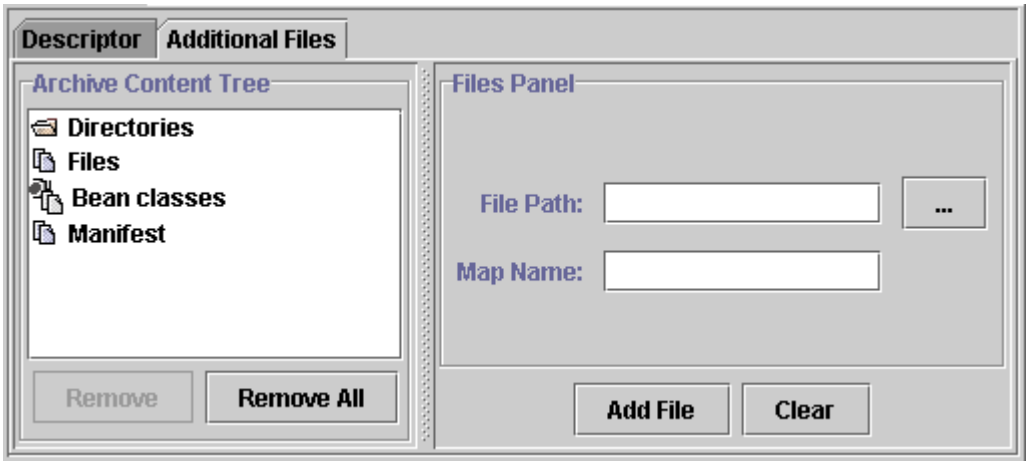
The “Descriptor” tab provides the following information about the JAR file:

- Display Name – a short name that is intended to be displayed by tools that use this JAR
- Small Icon – describes the full path to a small (16 x 16) icon image. When specified, the icon appears in the “Small Icon” pane.
- Large Icon – describes the full path to a large (32 x 32) icon image. When specified, the icon appears in the “Large Icon” pane.
- Set default – restores the default icon settings
- Description for the JAR – a descriptive text about the selected JAR file



The “Descriptor” Tab

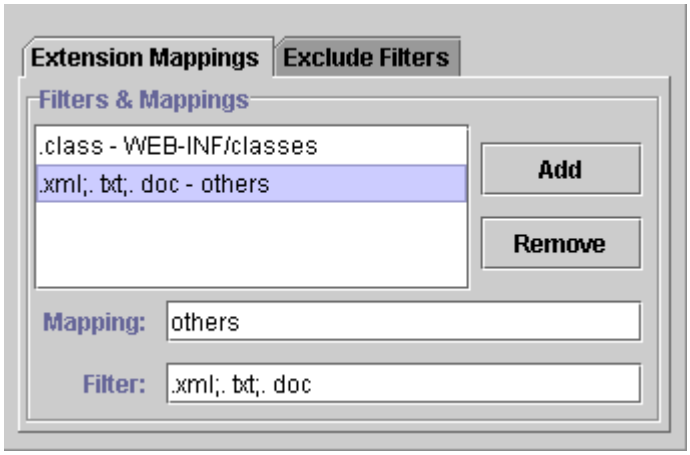
The “Additional Files” tab displays a list of all additional files, apart from the EJB components, that must be included in the JAR.



The “Additional Files” Tab

- “Archive Content Tree” pane – displays a list of directories, files, Bean classes and the manifest file added to the archive. A selected element can be removed using the “Remove” option. “Remove All” removes all components added to the “Archive Content Tree.” The content of the right-hand pane differs according to the selected component from the “Archive Content Tree” pane.
- “Directories Panel” – is displayed when a directory is selected from the “Archive Content Tree” pane. The selected folder is displayed with its name, and the full path is displayed in “Directories Panel.” This panel contains the following features
  - “Files In Directory” – if the “Show with filters” box is set, the “Files In Directory” pane displays all files from the selected directory filtered with the specified filters. These files are added to the JAR file.
  - “Extension Mappings” tab – specifies the mapping of some types of files you want to add to the JAR file. This option helps to increase understanding and improve file structuring of the Java archive file. The “Filters & Mappings” pane displays a list of all mappings and filters, defined for the selected directory using the “Mapping” and “Filter” fields. “Mapping” specifies the directory where the filtered files are to be added. If the “Mapping” field points to an already added JAR file directory, the filtered files are added to the files in that directory. If the “Mapping” field points to a non-existent directory, a new directory with this name is created in the JAR file. The “Filter” field specifies the extension by which the files are filtered before being added to the directory specified in the “Mapping” field. When specifying more than one filter, use a semicolon to separate them. Use “Add” and “Remove” to change the list in the “Filters & Mappings” pane.

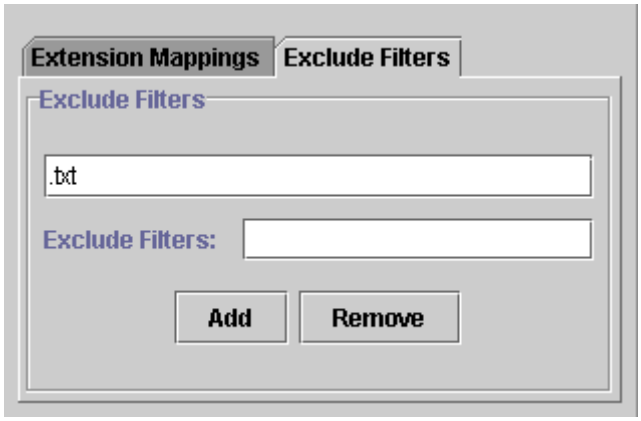
Note: When a mapping is used for a file, only the beginning of the path is changed. The structure of the rest of the path remains the same.



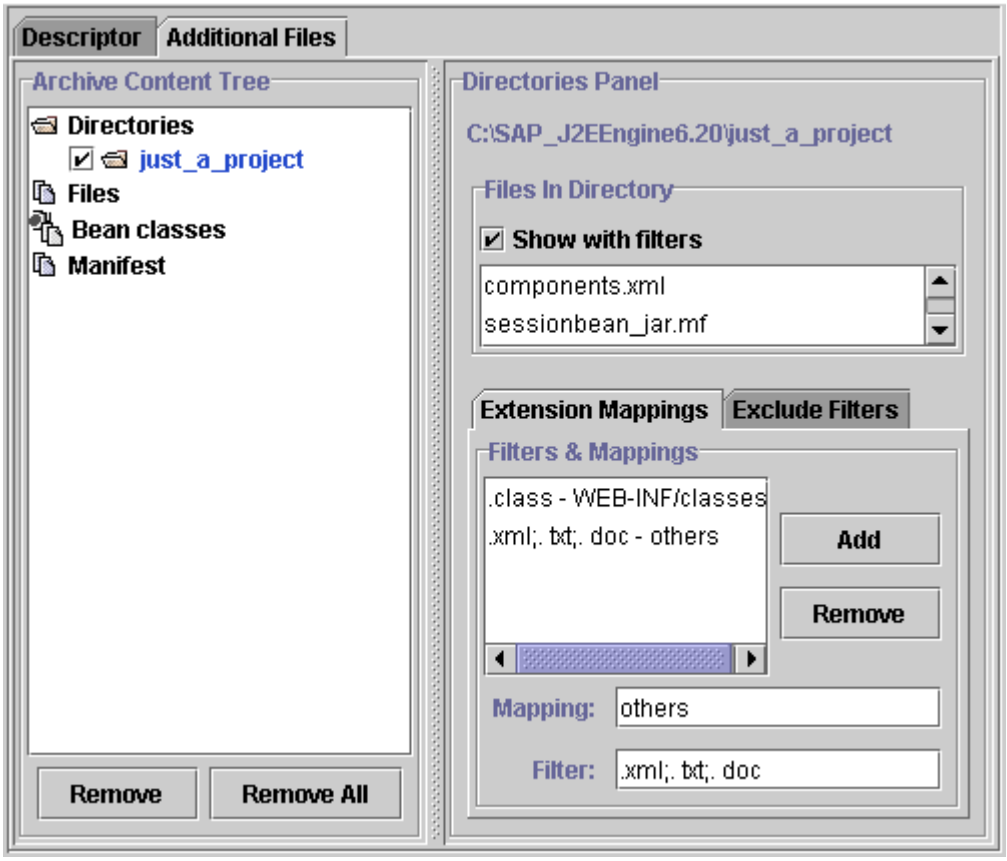
The “Extension Mappings” Tab

- “Exclude Filters” – displays a list of all filters for files to be excluded from the selected directory. When specifying more than one filter, use a by semicolon to separate them. Use “Add” and “Remove” to change the list in the “Exclude Filters” pane.

Note: Directories that are not filtered by include or exclude filters, are added to the project with their original structure.



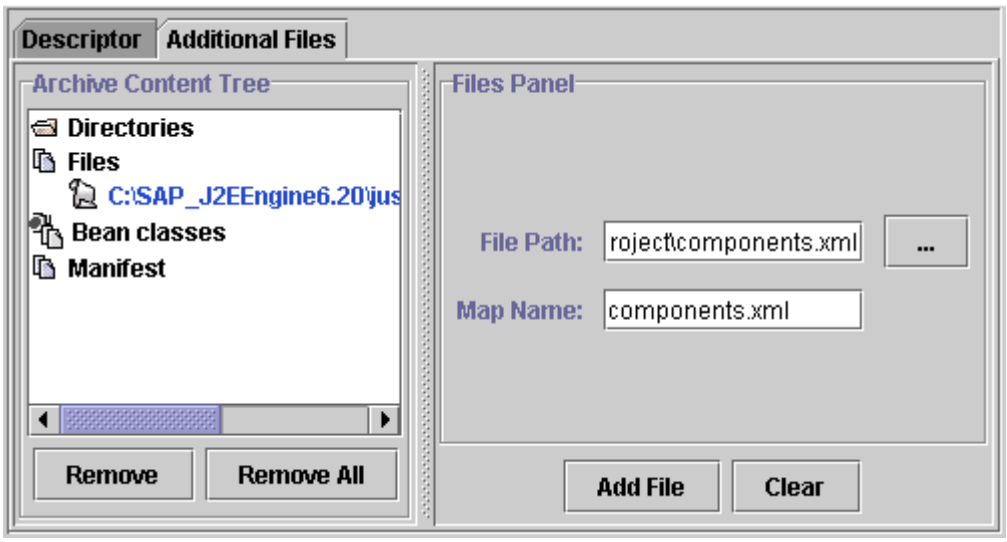
The “Exclude Filters” Tab



The “Directories” Panel

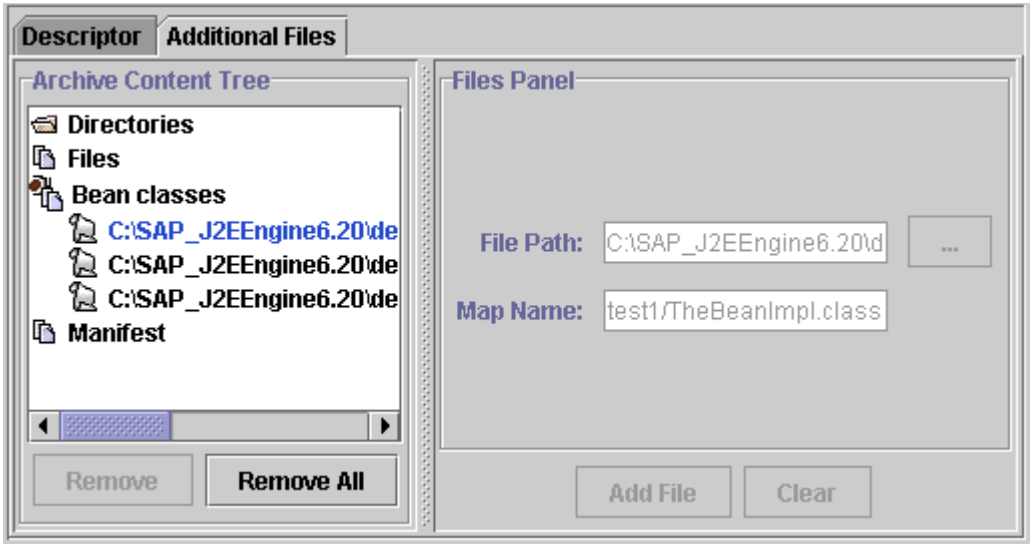
- “Files Panel” is displayed when the “Files” component is selected from the “Archive Content Tree” pane. This feature is useful when adding not a whole directory but a single file. It contains the following fields:
  - “File Path” – specifies the path to the file to be added to the archive
  - “Map Name” – specifies the name the file uses in the archive

Note: A **_jar.mf* file, where * is the name of the JAR file, is created and added to the archive automatically with *META-INF/MANIFEST.MF* mapping name.



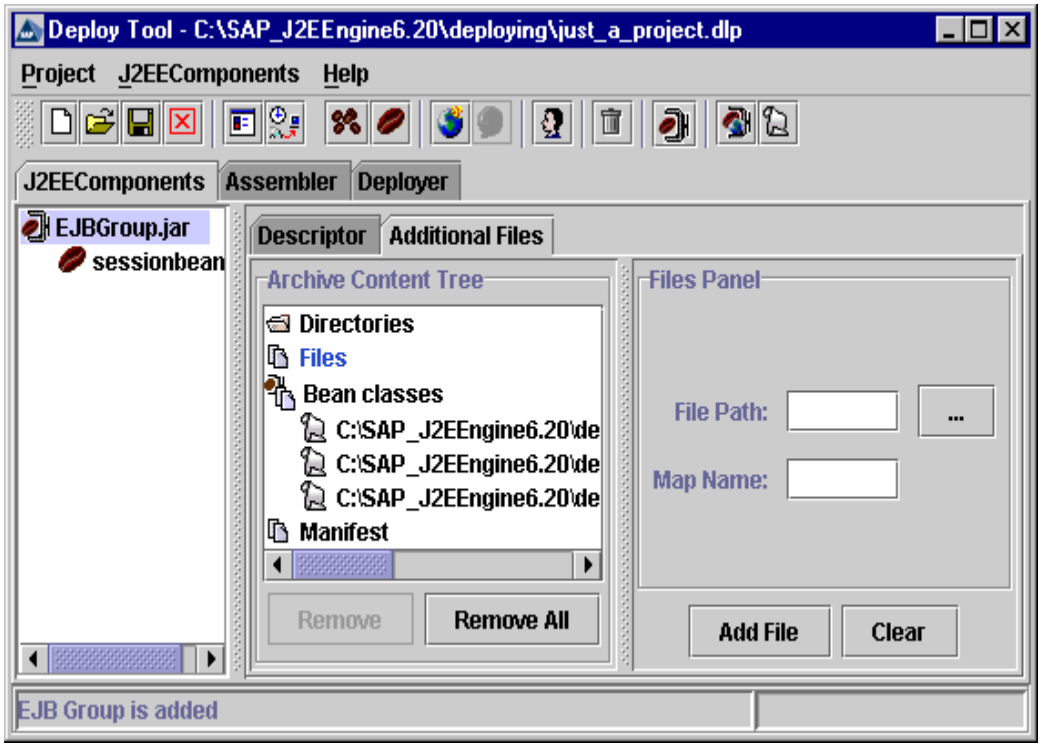
The “Files Panel”

- “Bean classes” consist of the classes of Bean files already added to the EJBGroup. The content of the “File Path” and “Map Name” fields is generated automatically and cannot be modified from this panel. For further information, refer to the *Add EJB* section below.



The “Bean classes”

- “Manifest Panel” – according to the J2EE™ Specification, all Java archive files have a manifest file. The “Manifest Panel” enables you to edit the manifest file. This is not recommended for EJB JAR and WAR files. You must set information of type: `MAIN-CLASS: <ClassName>` for the Java client archives for the main class to be started directly from the Java client archive. When a manifest file is added, it appears as a tree node in the “Files” tree. The “Map Name” field value is *META-INF/MANIFEST.MF*.

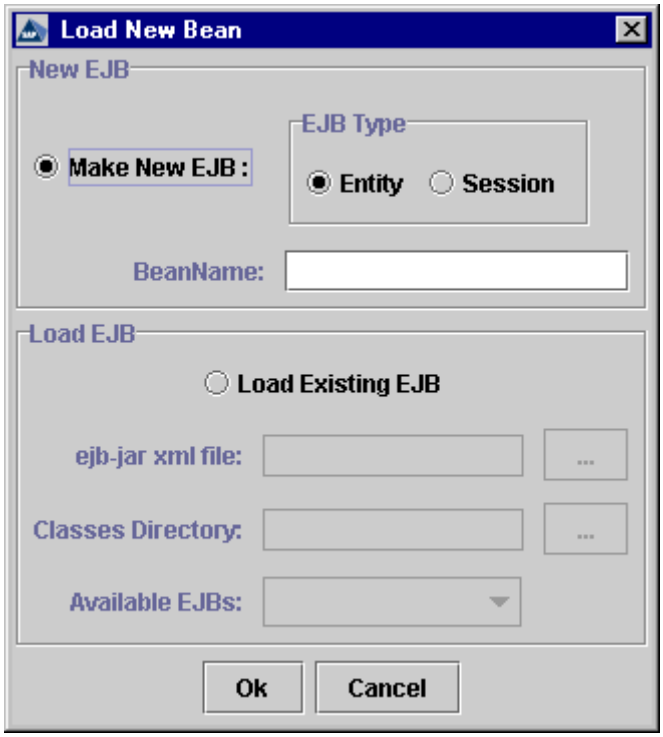


"Additional Files" Tab

Add EJB

The *Add EJB* command or on the toolbar – adds an EJB to the selected EJB Group (that is, to the selected JAR). A new or existing Bean can be added. Two options are available in the dialog box that appears:

- "Make New EJB" – creates a new EJB. Enter the EJB type (session or entity) and the Bean name. After creating the EJB, its Home interface, Remote interface, and Bean's class must be specified. These properties consist of the fully qualified Java class name of the EJB, beginning with the package name. The path to the package directory must be set in the *Project→Options→Classpath* menu.
- "Load Existing EJB" – loads a created EJB. Specify the enterprise bean deployment descriptor in the "ejb-jar xml file" field and the Java "Classes Directory." When the correct XML file is selected, you can choose the Bean by name from the described Beans in the XML file by using the "Available EJBs" field. If an incorrect XML file is specified, the field "Available EJBs" field is disabled. If some of the components are not specified correctly, the EJB is not added to the project.



The “Load New Bean” Window

EJBs can be added to a JAR with “New EJB” and “Load EJB” as well.

After adding the EJB to the JAR, select the EJB from the left-hand pane and view its properties. The basic properties of an EJB can be edited using six tabs – “General,” “Security,” “Transaction,” “Environment,” “References,” and “Additional.”

General Tab

This tab provides the following information about each EJB:

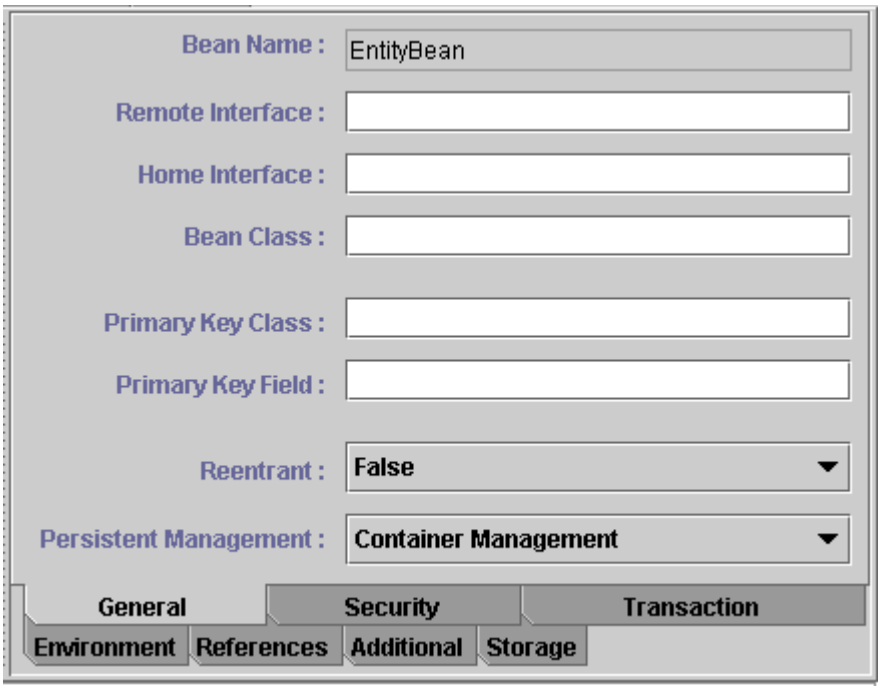
- “Bean Name” – specifies the EJB’s name used for lookup by clients
- “Remote Interface” – specifies the fully qualified EJB remote interface name. If the specified Java class directory is incorrect, the EJB cannot be loaded and an error message appears. If the EJB has been added properly, its remote interface appears as a “Bean classes” tree node in the “Archive Content Tree” pane in the “Additional Files” tab of the parent EJB Group.
- “Home Interface” – the fully qualified EJB home interface name. If the specified Java class directory is incorrect, the EJB cannot be loaded and an error message appears. If the EJB has been added properly, its home interface appears as a “Bean classes” tree node in the “Archive Content Tree” pane in the “Additional Files” tab of the parent EJB Group.
- “Bean Class” – specifies the fully qualified EJB class name. If the specified Java class directory is incorrect, the EJB cannot be loaded and an error message appears. If the EJB has been added properly, its remote interface appears as a “Bean classes” tree node in the “Archive Content Tree” pane in the “Additional Files” tab of the parent EJB Group.

Note: The information in the fields above must be specified correctly for the project to proceed. If one or more components are incorrect an error message box appears

when Deploy Tool tries to load the EJB. It checks whether the data is correct with every switch from tab to tab and from frame to frame.

For entity EJBs only, the following fields must be specified in the “General” tab:

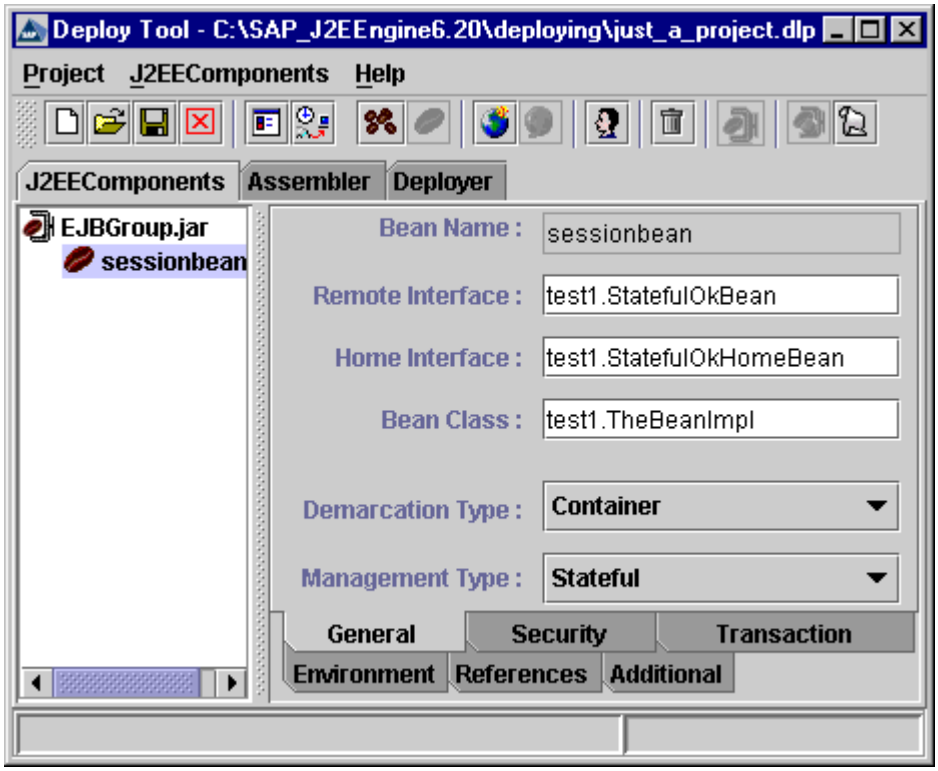
- “Primary Key Class” – defines the primary key. The primary key class must be specified both for entity Beans with bean-managed and container-managed persistence. For example, the Primary Key class can be `java.lang.Object`. It must be specified before deployment – that is, in the “Assembler” tab.
- “Primary Key Field” – specifies the “Primary Key Field” name. The primary key field is required for entity Beans with container-managed persistence. If you define the Primary Key class, the “Primary Key Field” must be left empty.
- “Reentrant” – if it is set to `true` the EJB can be accessed recursively
- “Persistent Management” – specifies the entity EJB type; container-managed or bean-managed.



The “General” Tab for Entity EJBs

For session Beans only, the following fields must be specified in the “General” tab:

- “Demarcation Type” – specifies whether the demarcation transaction is performed by the Bean or by the Container
- “Management Type” – specifies whether the session Bean is stateful or stateless

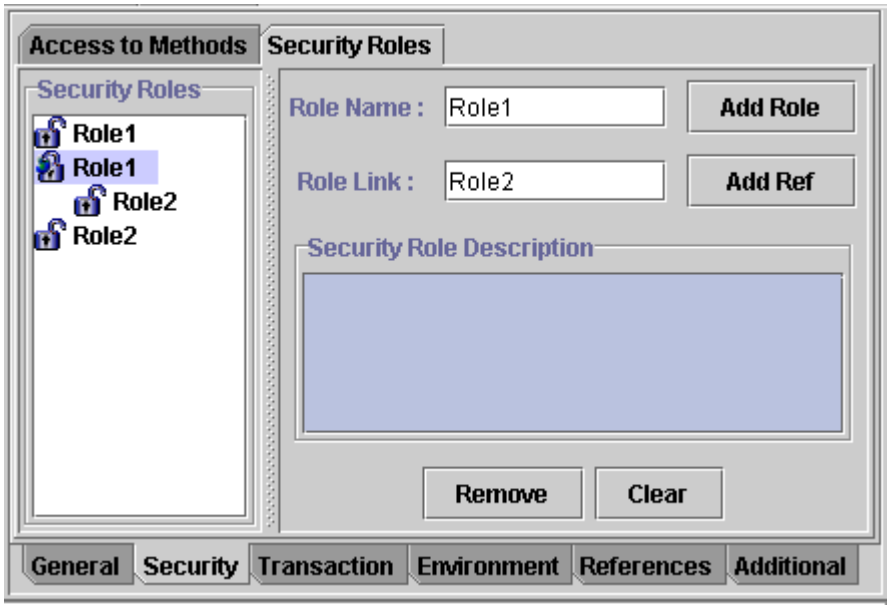


The “General” Tab for Session EJBs

Security Tab

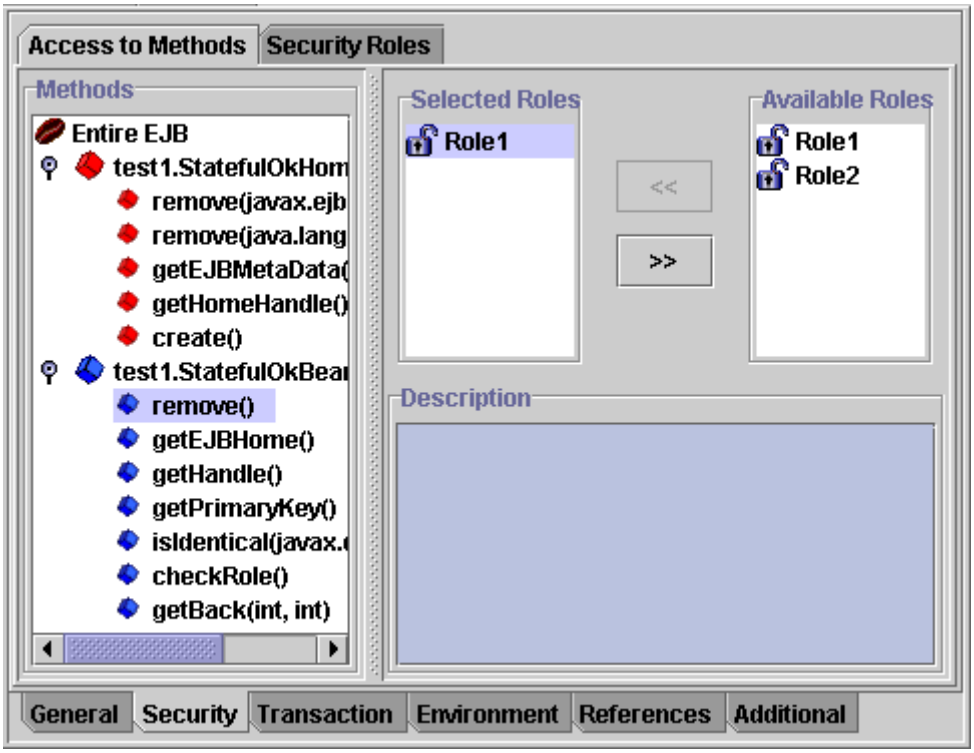
One or more security roles can be defined for each EJB in the Security tab. Security roles define the client access to the deployed EJBs remote interface methods. The security properties of an EJB can be edited using the tabs “Security Roles” and “Access to Methods,” as follows:

- “Security Roles” tab – a security role can be defined by specifying the following parameters in the “Security Roles” tab
  - “Role Name” field – specifies the name of the security role to be added to the “Security Roles” list
  - “Role Link” field – specifies the name of the link to a defined security role, or the name of the security role reference. An existing security role can be added as a role reference. A Security role can have only one security role reference. If a security role is not specified when generating the J2EE components, it must be specified in the *Assembler→Descriptor→Security* subtab.
  - “Security Roles” list – displays a list of available security roles and security role references
  - “Security Role Description” textbox – contains a descriptive text about the security role or security role reference.



The “Security Roles” Tab from “Security” Tab of an EJB

- “Access to Methods” tab – defined security roles can be bound with access rights to a single method or an entire EJB (that is, to all of its methods). The “Access to Methods” tab contains a list of mnemonic names of all defined security roles. The following fields of the “Access to Methods” tab are available
  - “Methods” pane – displays a list of all available methods in the remote and home interfaces of the EJB. You can specify certain security roles for each method, or for the entire EJB.
  - “Selected Roles” pane – displays a list of roles, chosen from the “Available Roles” list, and assigned to the selected method or entire Bean. You can add remove roles from the “Selected Roles” list using the direction arrows.
  - “Available Roles” pane – displays a list of all security roles that are added by the user in the “Security Roles” tab
  - “Description” pane – contains a descriptive text about the purpose of the security role displayed in the “Selected Roles” list

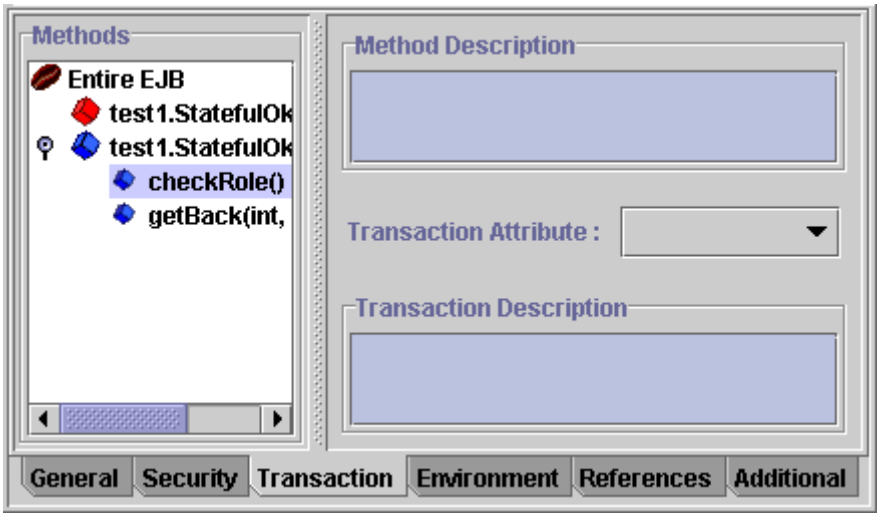


The “Access to Methods” Tab from the “Security” Tab of an EJB

Transaction Tab

This tab specifies the value of the transaction attributes of the EJB’s home and remote interface methods. These attributes are specified only for EJBs that require container-managed transaction demarcation:

- “Methods” pane – displays a list of all EJB home and remote interface methods
- “Method Description” pane – displays a descriptive text about the selected method
- “Transaction Attribute” – specifies how the container must manage the transaction boundaries when delegating a method invocation to an EJB’s business method. The possible values are `NotSupported`, `Never`, `Required`, `Supports`, `RequiresNew` and `Mandatory`. For session beans that implement session synchronization, the possible values are `Required`, `RequiresNew` and `Mandatory`.
- Transaction Description – displays a descriptive text about the transaction attribute



The “Transaction” Tab of an EJB

Environment Tab

The “Environment” tab enables you to the EJB to be customized without accessing or changing its source code. Each EJB defines its own set of environment entries. All EJBean instances within the same home share the same environment entries – the environment entries are not shared with other Beans. EJB instances are not allowed to modify the Bean’s environment at runtime. All EJB entries must be defined when generating J2EE components. The value (but not the name) is subject to change on the “Assembler” and “Deployer” tabs.

The following values for the EJB environment entries can be set for each home interface:

- “Name” – specifies the environment entry name
- “Type” – specifies the fully qualified Java type of the environment entry value, expected by the Bean’s code. The possible values are: `java.lang.Boolean`, `java.lang.String`, `java.lang.Integer`, `java.lang.Float` and `java.lang.Double`.
- “Value” – specifies the current value of the property. It must be a valid string, according to the environment entry type.
- “Description” – a descriptive text for the environment entry

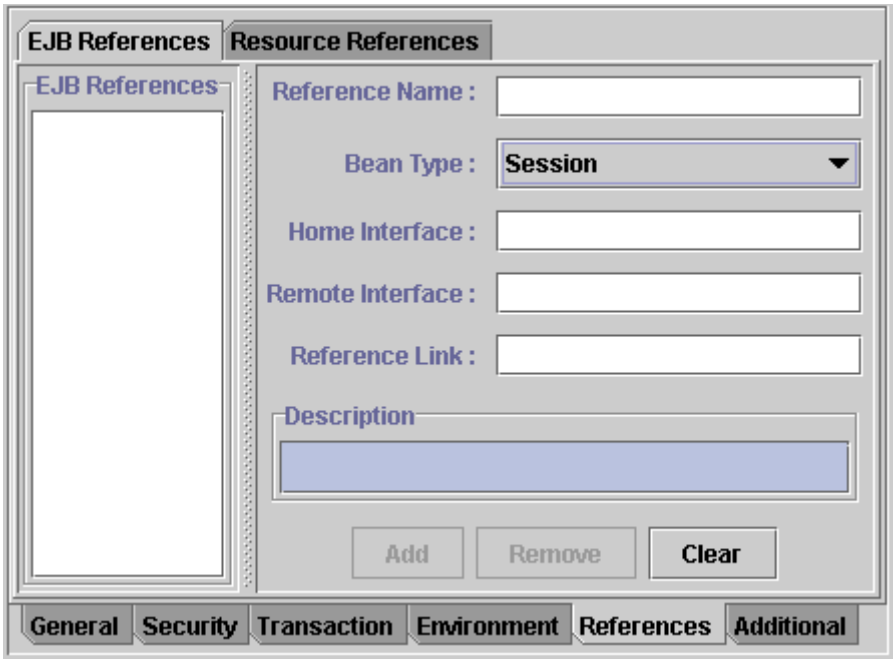
All EJB environment entries are displayed in a table containing their “Environment Name,” “Property Type,” and “Environment Value.”

The “Environment” Tab

References Tab

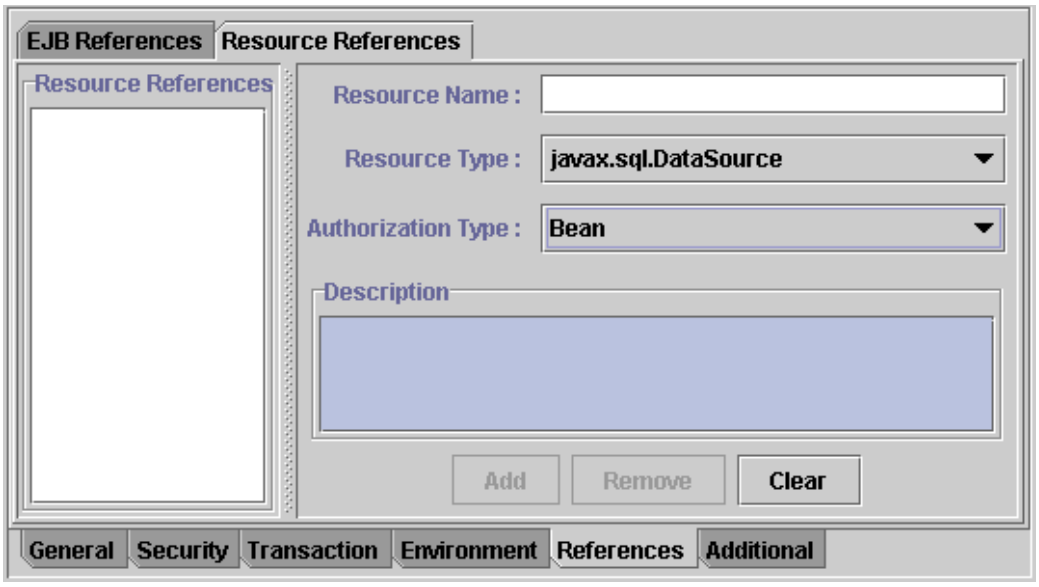
The “References” tab enables you to link EJBs to other Beans or data sources. This tab contains two subtabs:

- “EJB References” tab – consists of the following fields:
  - “EJB References” – displays a list of all EJB’s references to other Beans. The list contains the names of the referenced Beans.
  - “Reference Name” – the name the EJB uses to refer to the referenced Bean
  - “Bean Type” – specifies the type of the referenced Bean – entity or session
  - “Home Interface” – specifies the fully qualified home interface name of the referenced Bean
  - “Remote Interface” – specifies the fully qualified remote interface name of the referenced Bean
  - “Reference Link” – specifies the lookup name of the referenced Bean
  - “Description” – a descriptive text about the reference



The “EJB References” Tab

- “Resource References” – contains the following fields
  - “Resource Name” – the name of the environment entry used in the EJB source code
  - “Resource Type” – the type of the resource manager connection factory the Bean code expects. It can be one of the following types `javax.sql.DataSource`, `javax.jms.QueueConnectionFactory`, `javax.jms.TopicConnectionFactory`, `javax.mail.Session`, `javax.resource.cci.ConnectionFactory` and `java.net.URL`.
  - “Authorization Type” – specifies the responsibility for configuring the sign-on information of the resource manager. It can be either Bean or Container.
  - “Description” – a descriptive text about the reference



The “Resource References” Tab

Additional Tab

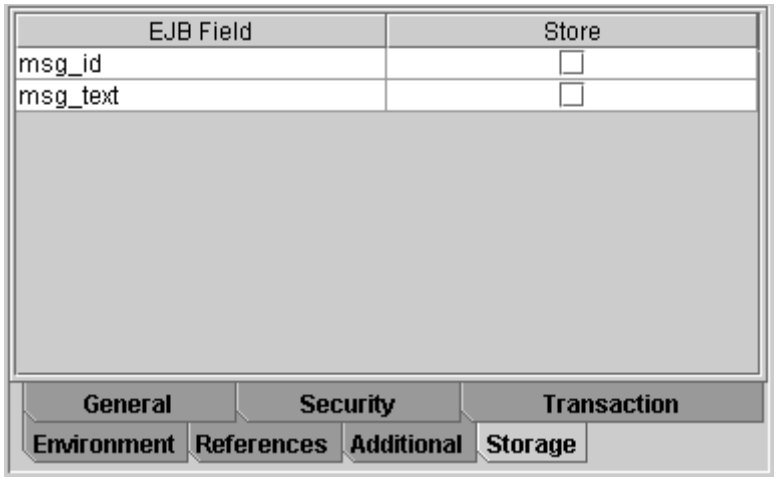
The “Additional” tab provides general information about the EJB.

- “Display Name” – specifies the EJB name in the JAR
- “Small Icon” – specifies the full path to the small (16 x 16) icon image used for the EJB. When specified, the icon appears in the “Small Icon” pane.
- “Large Icon” – specifies the full path to the large (32 x 32) icon image used for the EJB. When specified, the icon appears in the “Large Icon” pane.
- “Set default” – restores the default icon settings
- “Description” – a descriptive text about the component

Storage Tab

The “Storage” tab is available for entity EJBs with container-managed persistence only.

This tab enables you to define which “EJB Field” must be stored in the database.

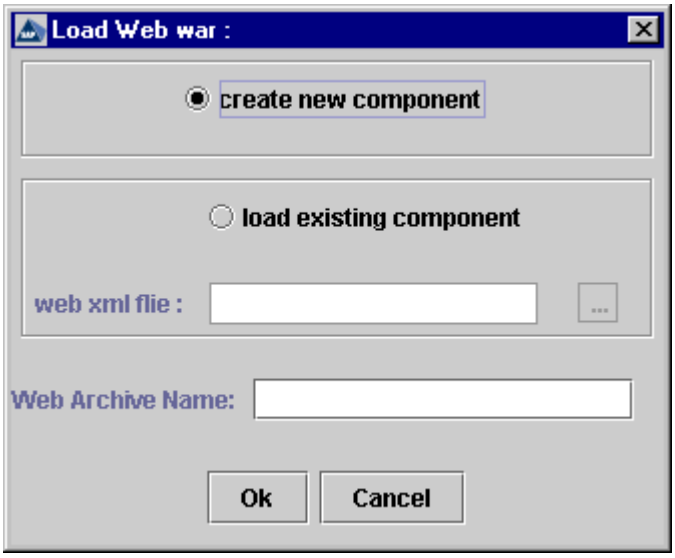


The “Storage” tab

Add Web

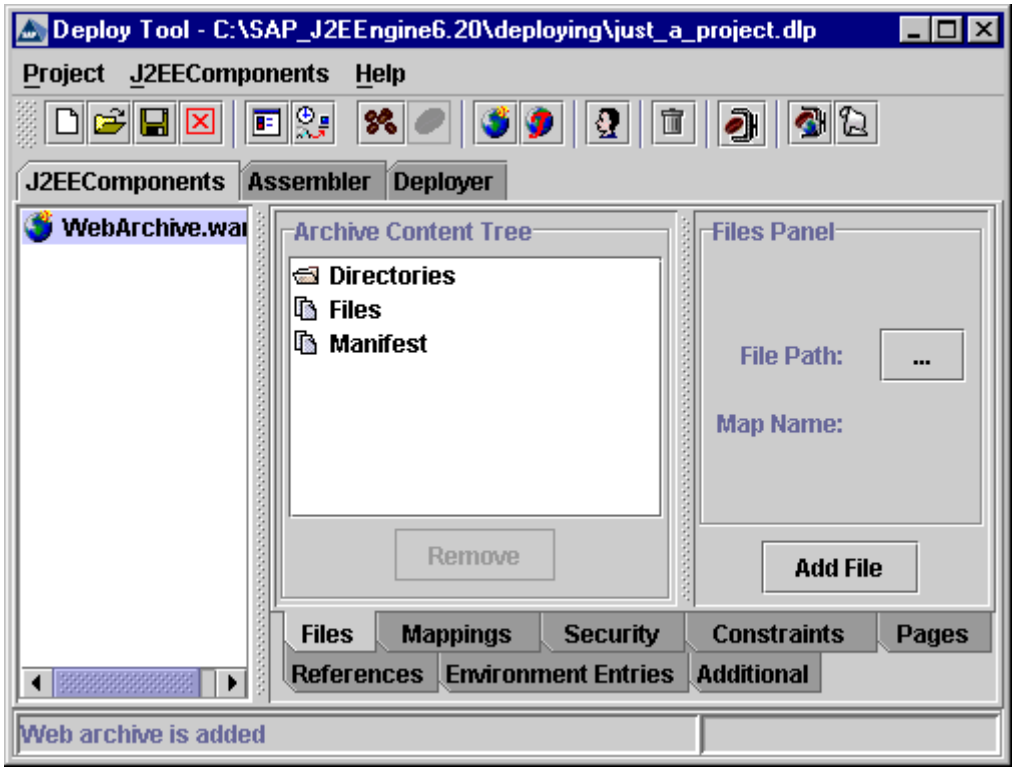
The *Add Web* command, or  on the toolbar – adds a new or existing Web group to the project. If the Web Archive exists, you can load it from an existing XML file. Executing this command opens a dialog box that provides the following options:

- “Create new component” – creates a new Web Archive. The “Web Archive Name” field must be specified.
- “Load existing component” – adds a created Web Archive, containing Servlet or JSP and other additional files, to the current project. The “XML File” and “Web Archive Name” fields must be filled in.



The Add a Web Archive

The basic Web Archive properties are specified in eight tabs – “Files,” “Mappings,” “Security,” “Constraints,” “Pages,” “References,” “Environment Entries,” and “Additional.”

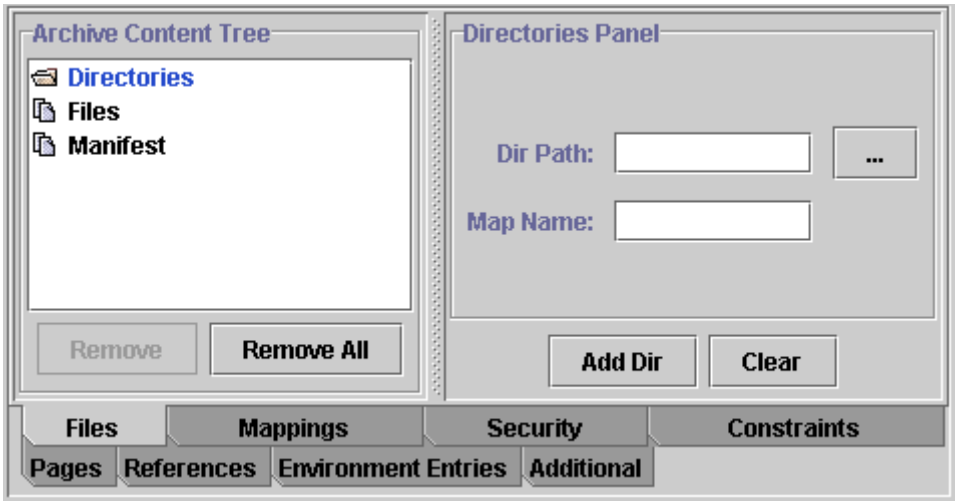


The Basic Web Archive Properties are Specified in Eight Tabs

Files Tab

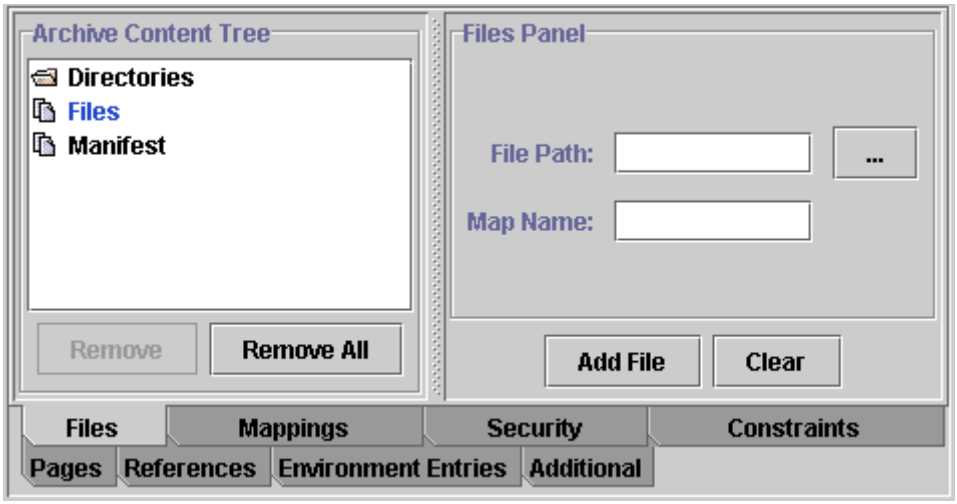
The “Files” tab appends files or whole directories to a Web Archive file. The tab contains the following options:

- "Archive Content Tree" pane – displays a list of directories, files, and the manifest file added to the archive
  - To add directories select "Directories" from the "Archive Content Tree" pane. Select "Add Dir," and the "Select Directory" dialog box appears. Browse to the desired directory, and choose "Ok." When a directory is appended, all its files and subdirectories are added to the WAR file. All files are specified by their relative paths. Servlets classes and the classes that are used by the Servlet are mapped automatically with "WEB-INF/classes." The directories appear as tree nodes in the "Archive Content Tree" pane. Using each directory checkbox, you can choose which directories to add to the project.



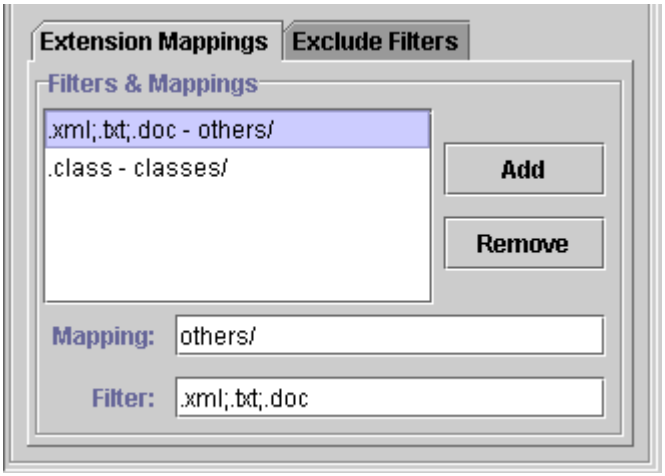
The "Directories Panel"

  - To add files, select "Files" from the "Archive Content Tree" pane. The "File Path" field in the "Files Panel" specifies the file path. The "Map Name" field specifies the name and type of the file to be appended to the WAR file. "Add File" confirms the adding. The files are added to the WAR file with their relative paths. The added files appear as tree nodes in the "Archive Content Tree" pane.



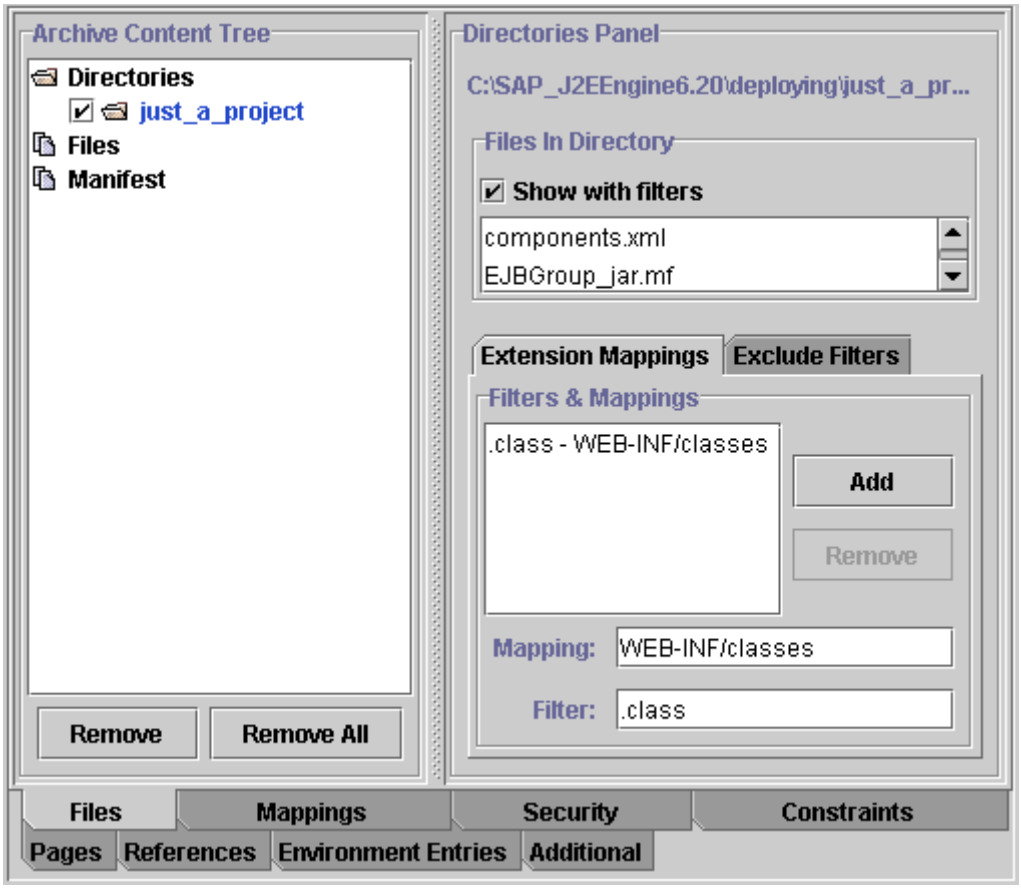
The "Files Panel"

- "Directories panel" – displayed when a directory is selected from the "Archive Content Tree" pane. The selected folder is displayed with its name – the full path is displayed in "Directories Panel." This panel contains the following features
  - "Files In Directory" – if the "Show with filters" box is set, the "Files In Directory" pane displays all files from the selected directory filtered with the specified filters. These files are added to the WAR file.
  - "Extension Mappings" tab – specifies the mapping of some types of files you want to add to the WAR file. This option helps to increase understanding and improve file structuring in the archive file. The "Filters & Mappings" pane displays a list of all mappings and filters, defined for the selected directory using the "Mapping" and "Filter" fields. "Mapping," specifies the directory where the filtered files are to be added. If the "Mapping" field points to an already added to the WAR file directory, the filtered files are added to the files in that directory. If the "Mapping" field points to a non-existent directory, a new directory with this name is created in the WAR file. The "Filter" field specifies the extension by which the files are filtered, and then added to the directory specified in the "Mapping" field. When specifying more than one filter, separate them by using a semicolon. Use "Add" and "Remove" to change the list in the "Filters & Mappings" pane.



The "Extension Mappings" Tab

  - "Exclude Filters" – displays a list of all filters for files to be excluded from the selected directory. When specifying more than one filter, separate them using a semicolon. Use "Add" and "Remove" to change the list in the "Exclude Filters" pane.

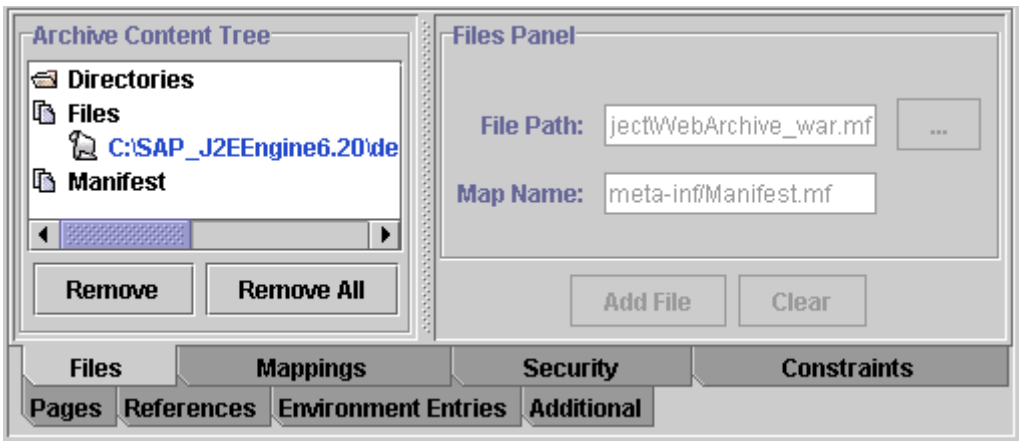


The “Directories Panel”

Note: Directories that are not filtered by include or exclude filters, are added to the project with their original structure.

- “Files Panel" is displayed when the “Files” component is selected from the “Archive Content Tree” pane. This feature is useful when not adding a whole directory but a single file. It contains the following fields
  - “File Path” – specifies the path to the file to be added to the archive
  - “Map Name” – specifies the name that the file will have within the archive.

Note: A **_war.mf* file, where * is the name of the WAR file, is created and added to the archive automatically with *META-INF/MANIFEST.MF* mapping name.



The “Files Panel”

- “Manifest Panel” – according to the J2EE™ Specification, all Java archive files have a manifest file. The “Manifest Panel” enables you to edit the manifest file. This is not recommended for EJB JAR and WAR files. A manifest file is added, and appears as a tree node in the “Files” tree. The Manifest file path is relative. The “Map Name” field value is *META-INF/MANIFEST.MF*.

Consider the following notes:

Note: Servlet deployment descriptors are saved in the WAR file under mapping name “WEB-INF” automatically.

Note: Each Servlet or JSP must have its own deployment descriptor that describes its classes.

Note: Servlets or JSPs (or both) are added to the Web Archive by executing the *J2EEComponents→Add Servlet/JSP→Add New...* menu for each Servlet or JSP. For further information, refer to the *Add Servlet/JSP* section.

Mappings Tab

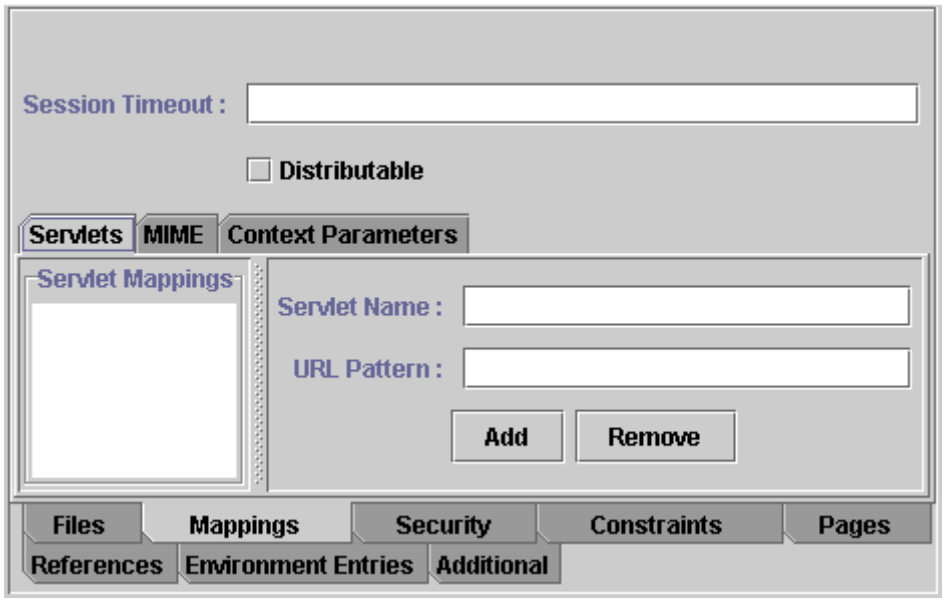
Servlet mappings, MIME mappings and context parameters can be specified in this tab. It contains the following components:

- “Cookie Configuration” – defines a Port and a Path. For a Port, you can set `APPLICATION`, `NONE` or to tape your own one. The default value is `APPLICATION`. For a Domain, you can set `SERVER`, `NONE` or to write your own one. The default value is `SERVER`. These values are editable.

Note: Each WAR may have only one Cookie.

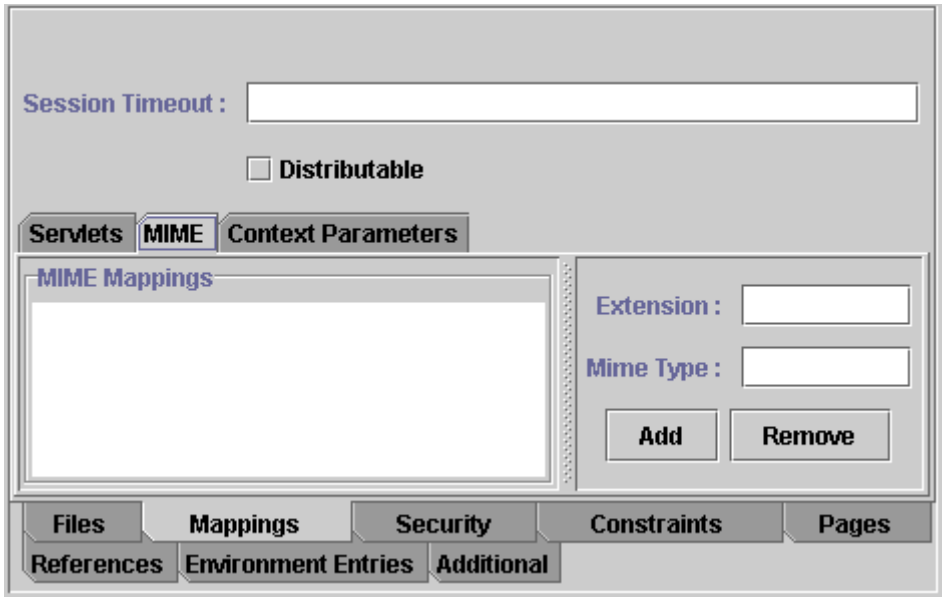
- “Session Timeout” – defines the session timeout period in minutes after which the session of the client expires. The value of this property must be of type integer. If the value is set to `1`, the session never expires.
- “Distributable” – indicates whether this Web application can be deployed in a distributed Servlet container
- “Servlets” tab – the Servlet container must use URL paths to map requests to Servlets. This is defined by specifying the following properties
  - Servlet Mappings – displays a list of all mappings between Servlets and a URL pattern
  - Servlet Name – specifies the name of the Servlet

- URL Pattern – specifies the URL pattern of the mapping



The "Servlets" tab

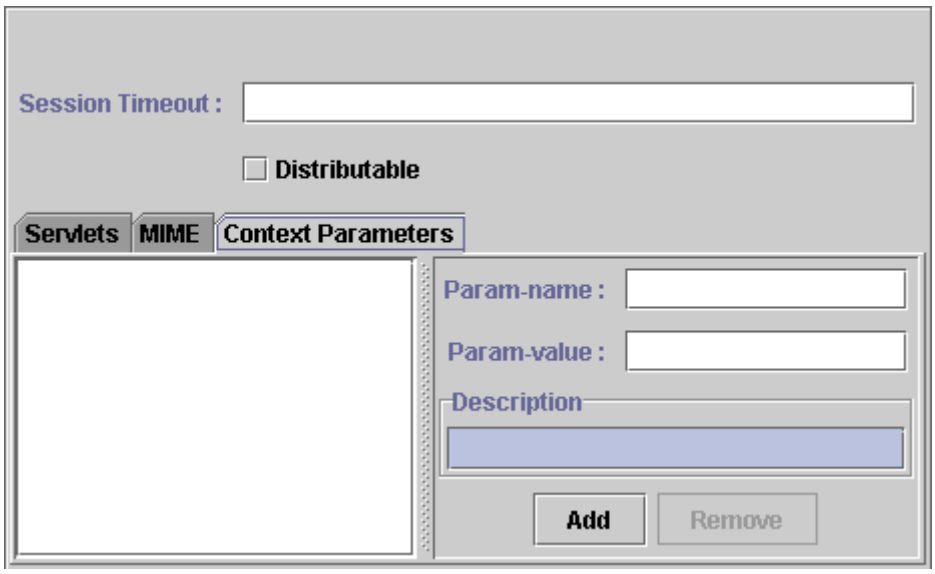
- "MIME" tab –defines mappings between extensions and MIME types. It contains the following options
  - "MIME Mappings" – contains a list of all MIME mappings
  - "Extension" – specifies a string describing the extension
  - "Mime Type" – specifies a defined MIME type



The "MIME" Tab

- "Context Parameters" tab – displays a list of the Web application's Servlet context initialization parameters. The Servlet context defines the Servlet view of the Web application it is running. This element must contain the declaration of the initialization parameters of the Servlet context. The following parameters can be specified
  - "Param-name" – specifies the name of the parameter

- “Param-value” – specifies the value of the parameter
- “Description” – describes a text about the parameter



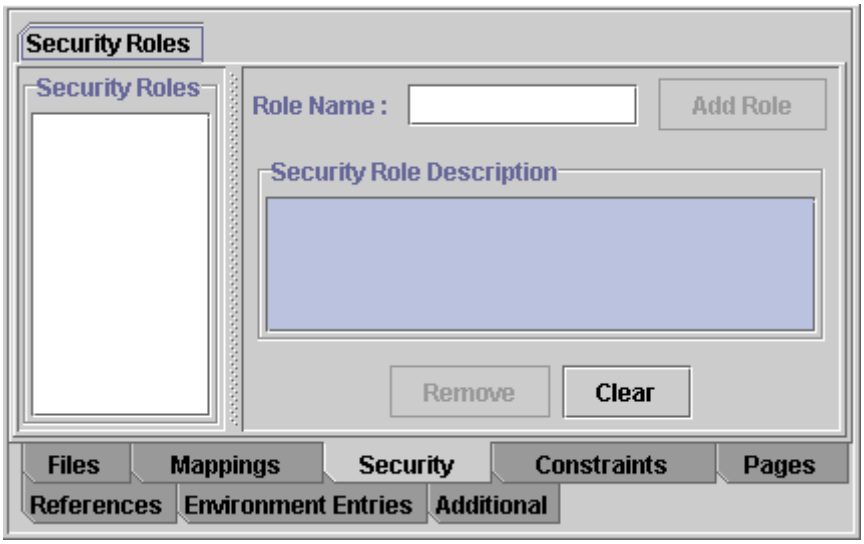
The “Context Parameters” Tab

Note: You must set the Servlet name and URL pattern in the WAR “Mappings” tab for each Servlet in the WAR.

Security Tab

The “Security” tab contains a set of properties that ensure the protection of the application. The tab consists of one subtab – “Security Roles.” It provides the following properties:

- “Security Roles” pane – displays a list of all defined security roles
- “Role Name” – specifies the name of the security role to be added to the “Security Roles” list
- “Security Role Description” – describes a text about the security role

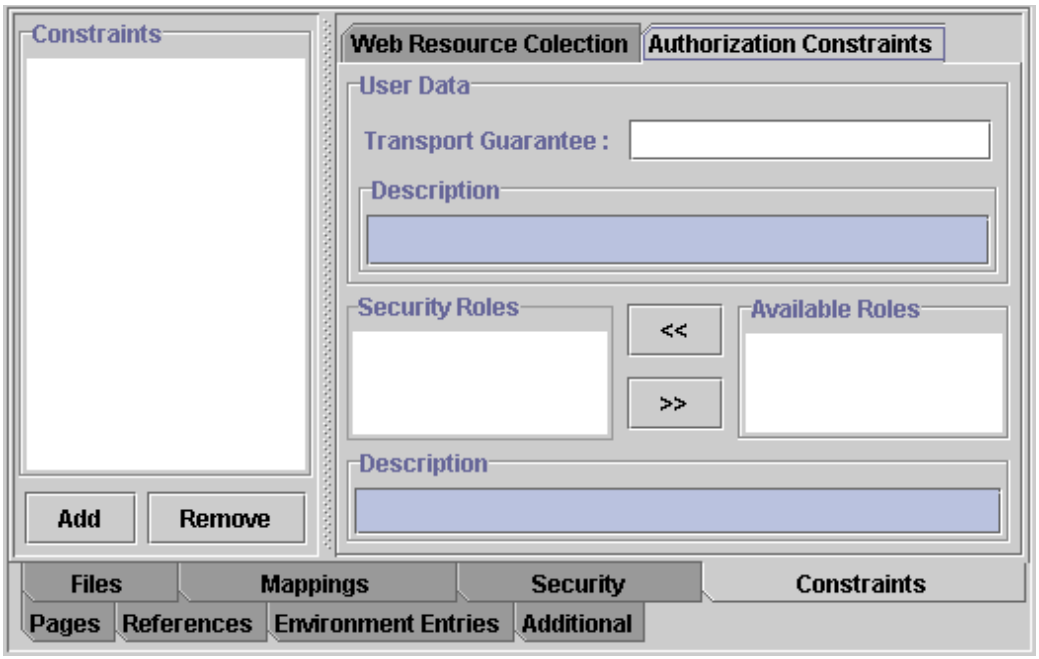


The “Security” Tab

Constraints Tab

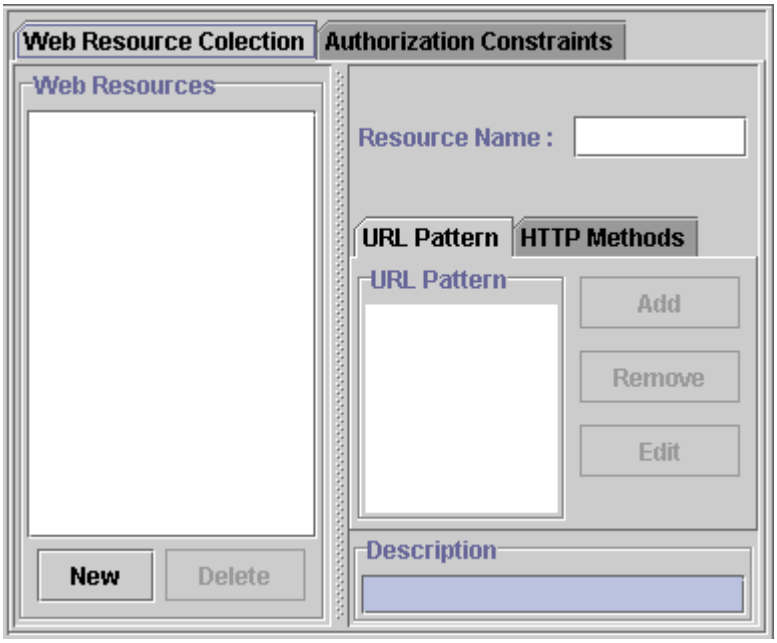
The “Constraints” tab is used to associate security constraints with one or more Web resource collections. The “Constraints” tab consists of the “Constraints” pane and two subtabs – “Web Resource Collection” and “Authorization Constraints.”

The “Constraints” pane displays a list of all defined constraints. You can add new constraints using “Add.” Use the two subtabs to modify the properties of the constraint.



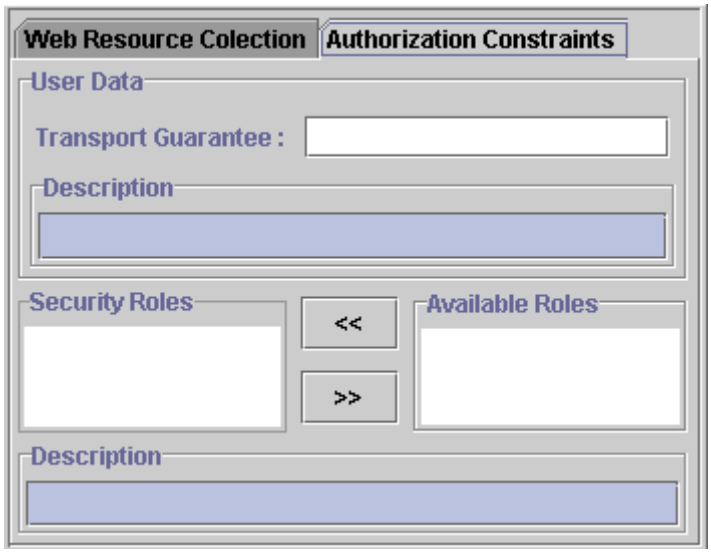
The “Constraints” Tab

- “Web Resource Collection” tab is used to define a subset of Web application resources and their HTTP methods, to which the selected security constraint applies. If no HTTP methods are specified the security constraint applies to all HTTP methods. The “Web Resource Collection” tab consists of several elements
  - “Web Resources” pane – displays a list of all defined Web resources
  - “Resource Name” – displays the name of the Web resource collection
  - “URL Pattern” tab – enables you to change the list of the mappings between the URL and the corresponding resource.
  - “HTTP Methods” tab – enables you to change the list of the HTTP methods
  - “Description” – displays a text of the selected Web Resource Collection



The “Web Resource Collection” Tab

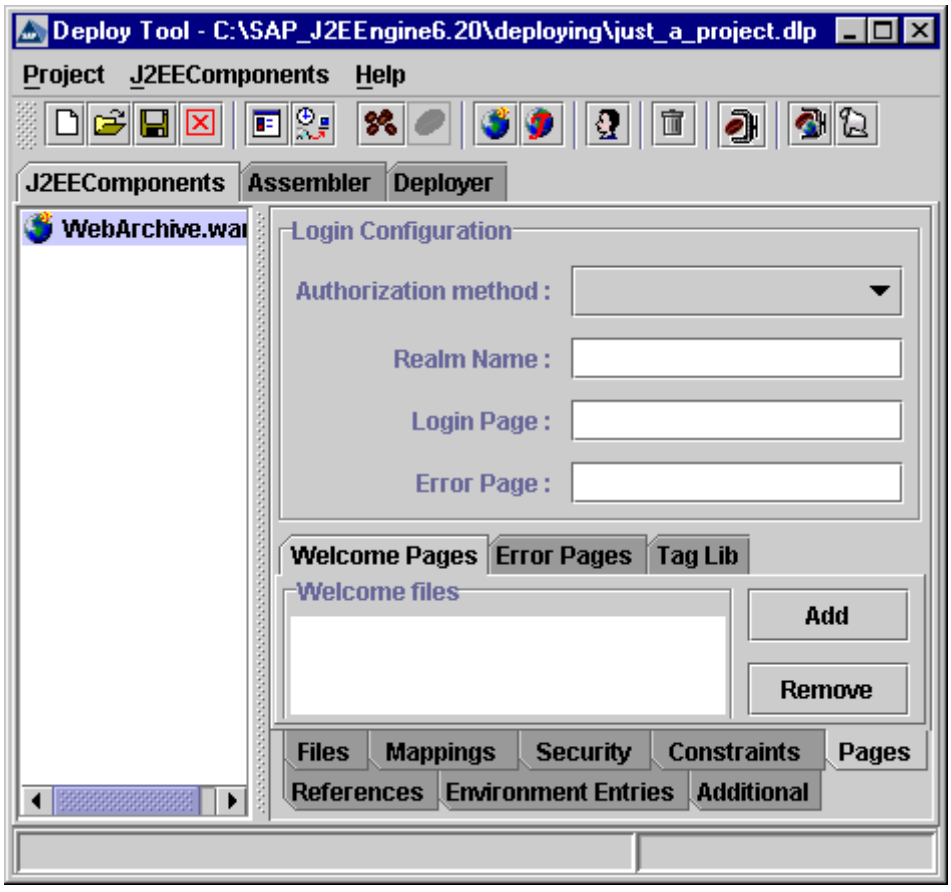
- “Authorization Constraint” tab – indicates the user roles that must be permitted access to this resource collection.
  - “User Data” pane – indicates how the data communication between the client and container must be protected.
  - “Transport Guarantee” – defines the communication protection between the client and the J2EE Engine. Possible values are `NONE` – protection is not required, `INTEGRAL` – data is sent in such a way that it can not be changed in transit, `CONFIDENTIAL` – data is sent in such a way as to prevent other entities observing the content of the transmission.
  - “Description” – displays a text of the transport guarantee component
  - “Security Roles” – if security roles are defined in the “Security” tab, they appear in the “Available Roles” subtab in the “Constraints” tab. They can be mapped to a definite constraint. In this case, they appear in the “Security Roles” subtab when the corresponding constraint is highlighted.
  - “Available Roles” – displays a list of all defined security roles
  - “Description” – displays a text of the corresponding security role



The “Authorization Constraints” Tab

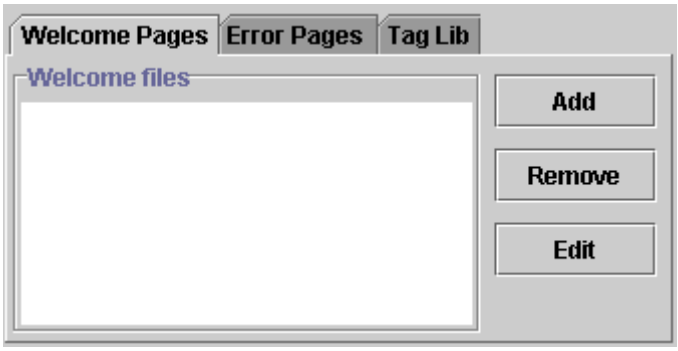
Pages Tab

- The “Login Configuration” pane is used to configure the Authorization method that must be used, the “Realm Name,” and the attributes that are needed for the login mechanism.
  - “Authorization method” – configures the authentication mechanism for the Web application. As a prerequisite for gaining access to any Web resources protected by an authorization constraint, you must authenticate using the configured mechanism. Legal values for this element are `BASIC` and `FORM`.
  - “Realm Name” – specifies the realm name to be used in the HTTP `Basic` authorization
  - “Login Page” – defines the location in the Web application where you can find the page used for login
  - “Error Page” – defines the location in the Web application where you can find the error page that is displayed when login is not realized



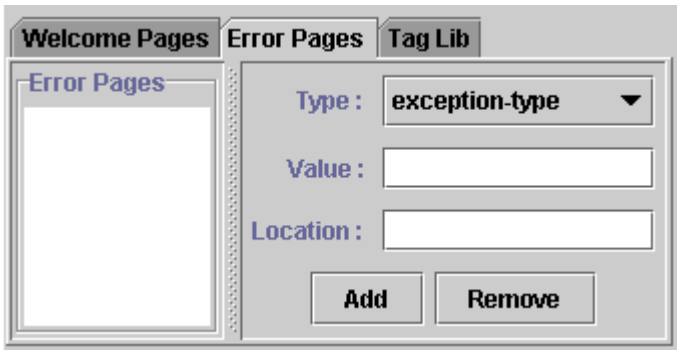
The “Pages” Tab

- “Welcome Pages” tab – changes and displays a list of welcome files. These are file names to use as default welcome files, such as *index.html*, and so on.



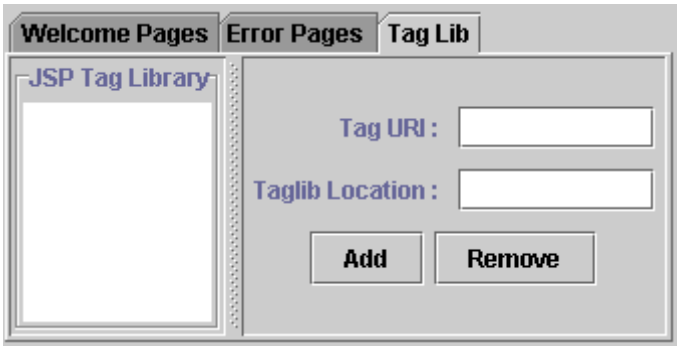
The “Welcome Pages” Tab

- Error Pages – displays a list of all mappings between an error code or exception type, and the Web application resource path
  - Type – specifies error code or exception-type types
  - Value – an HTTP error code (for example `404`) if the type is error code or a fully qualified class name of Java exception type
  - Location – specifies the resource location



The “Error Pages” Tab

- “Tag Lib” tab
  - “JSP Tag Library” – describes a JSP tag library
  - “Tag URI” – describes a URI, relative to the location of the Web. XML document, identifying a Tag Library used in the Web application
  - “Tag Lib Location” – specifies the location (as a resource relative to the root of the Web application) where you can find the Tag Library Description file for the tag library

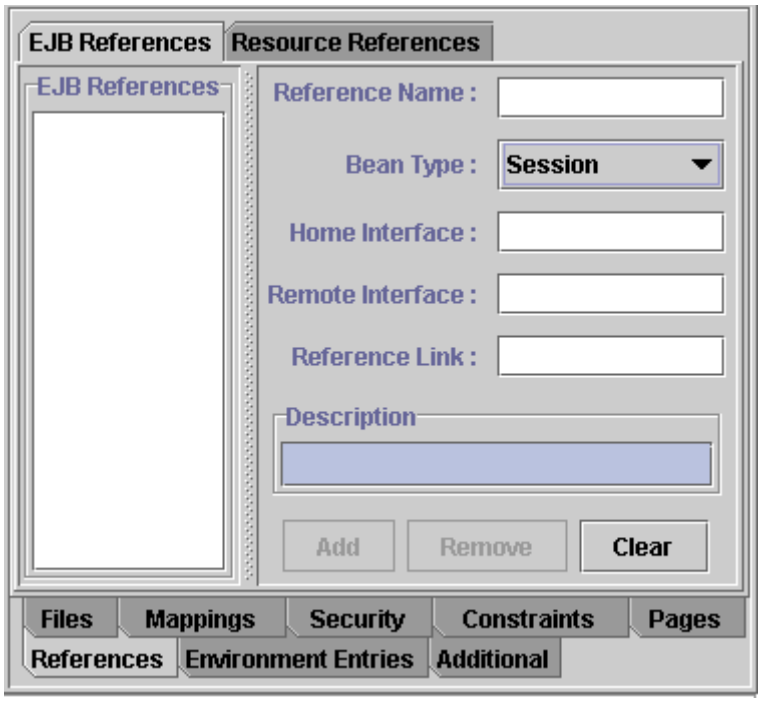


The “Tag Lib” Tab

References Tab

This tab is used to declare a reference to an enterprise Bean.

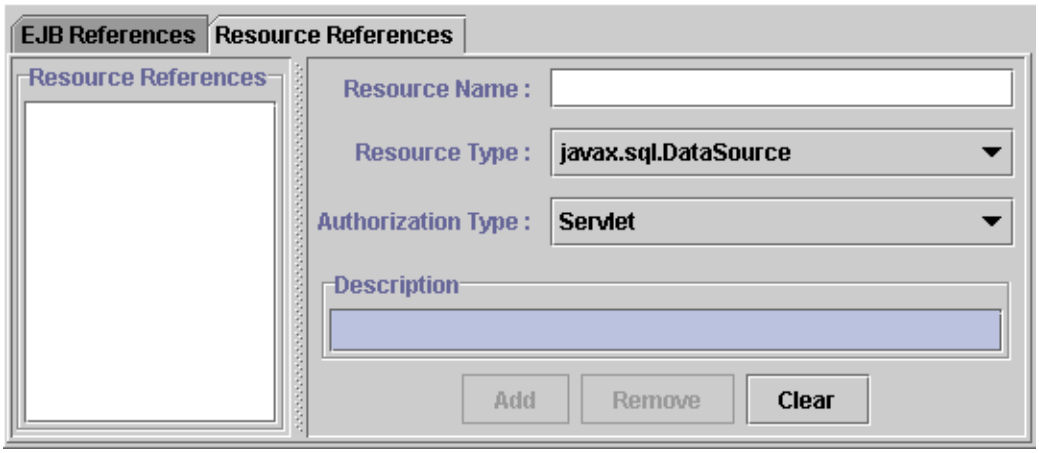
- "EJB References" tab
  - "EJB References" pane – displays a list of all EJB's references to other Beans. The list contains the names of the reference Beans.
  - "Reference name" – the JNDI name that the Servlet uses to get a reference to the Bean
  - "Bean Type" – specifies the type of the reference Bean – `Entity` or `Session`
  - "Home Interface" – specifies the fully qualified name of the EJB's home interface
  - "Remote Interface" – specifies the fully qualified name of the EJB's remote interface
  - "Reference Link" – specifies that an EJB reference be linked to an EJB in an encompassing J2EE application package. The value must be the name of an EJB in the J2EE application package that is, the name of the EJB in the JNDI.
  - "Description" – displays a text about the references



The "EJB References" Tab

- "Resource References" tab
  - "Resource References" – displays a list of all resource references to Beans
  - "Resource Name" – specifies the name of the environment entry
  - "Resource Type" – specifies the type of the resource manager connection factory that the Web code expects. It can be one of the following types `javax.sql.DataSource`, `javax.jms.QueueConnectionFactory`, `javax.jms.TopicConnectionFactory`, `javax.mail.Session`, `javax.resource.cci.ConnectionFactory` and `java.net.URL`.
  - Authorization Type – specifies the responsibility for configuring the sign-on information for the resource manager. It can be either `Servlet` or `Container`.

- “Description” – descriptive text about “Resource References”

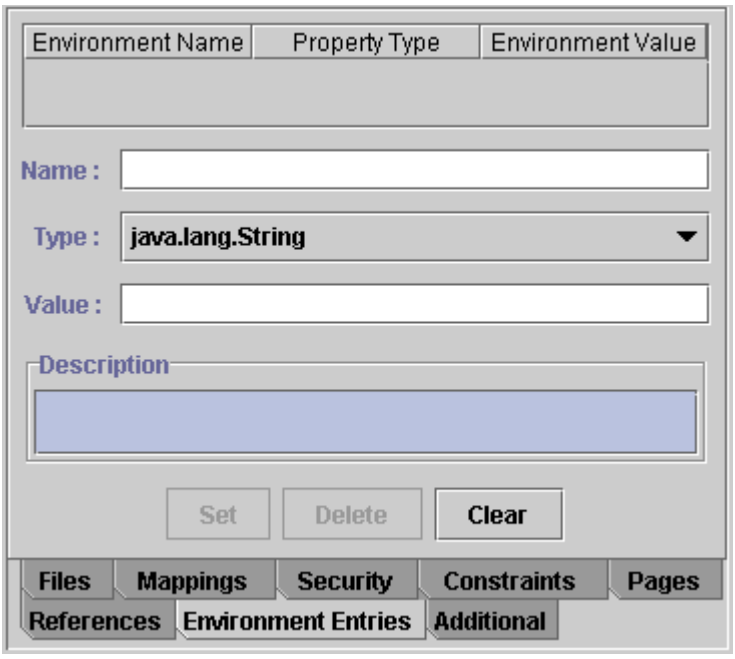


The “Resource References” Tab

Environment Entries Tab

The "Environment Entries" tab enables you to customize the EJBs without accessing or changing their source code. It displays a list of all specified environment entries with their “Environment Name,” “Property Type” and “Environment Value.”

- “Name” – specifies the environment entry name
- “Type” – specifies the fully qualified Java type of the environment entry value expected by the EJB code. The possible values are `java.lang.Boolean`, `java.lang.String`, `java.lang.Integer`, `java.lang.Float` and `java.lang.Double`.
- “Value” – displays the current value of the property. It must be a valid string, according to the environment entry type.
- “Description” – displays a text about the environment entry

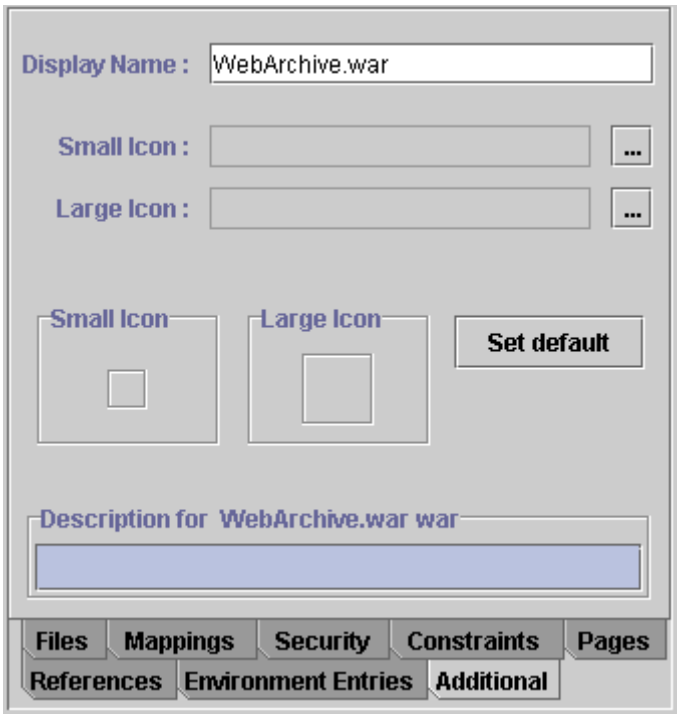


The “Environment Entries” Tab

Additional Tab

The “Additional” tab provides general information about the Web application:

- “Display Name” – specifies the WAR file path and name
- “Small Icon” pane – specifies the full path to the small (16 x 16) icon image used for the Web application. When specified the icon appears in the “Small Icon” pane.
- “Large Icon” pane – specifies the full path to the large (32 x 32) icon image used for the Web application. When specified the icon appears in the “Large Icon” pane.
- “Set default” button – restores the default icon settings
- “Description” – displays a short text about the WAR



The “Additional” Tab

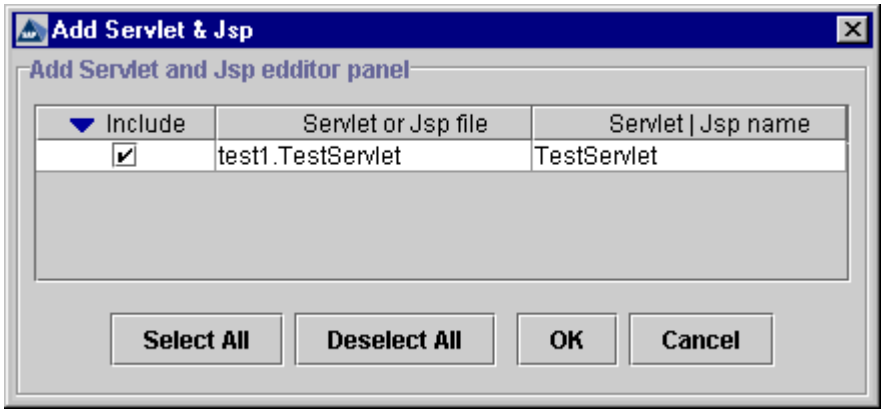
Add Servlet | JSP

You can add a Servlet or a JSP to a Web Archive using the *J2EEComponents->Add Servlet/JSP* menu or on the toolbar. There are two ways to add Servlet or JSP to a selected Web group:

- *J2EEComponents→Add Servlet/JSP→Add new...* or on the toolbar creates a new Servlet or JSP and adds it to the Web group. Enter the Servlet or JSP name in the dialog box that appears. You must specify the fully qualified Java class or JSP file.

"Add new..." window

- *J2EEComponents→Add Servlet/JSP→Add from files* – verifies all additional files that were added to the WAR's Archive Content Tree and provides a list of all possible Servlet or JSP files that can be created. Deploy Tool offers a name based on the Servlet or JSP files.



The "Add Servlet & Jsp" Window

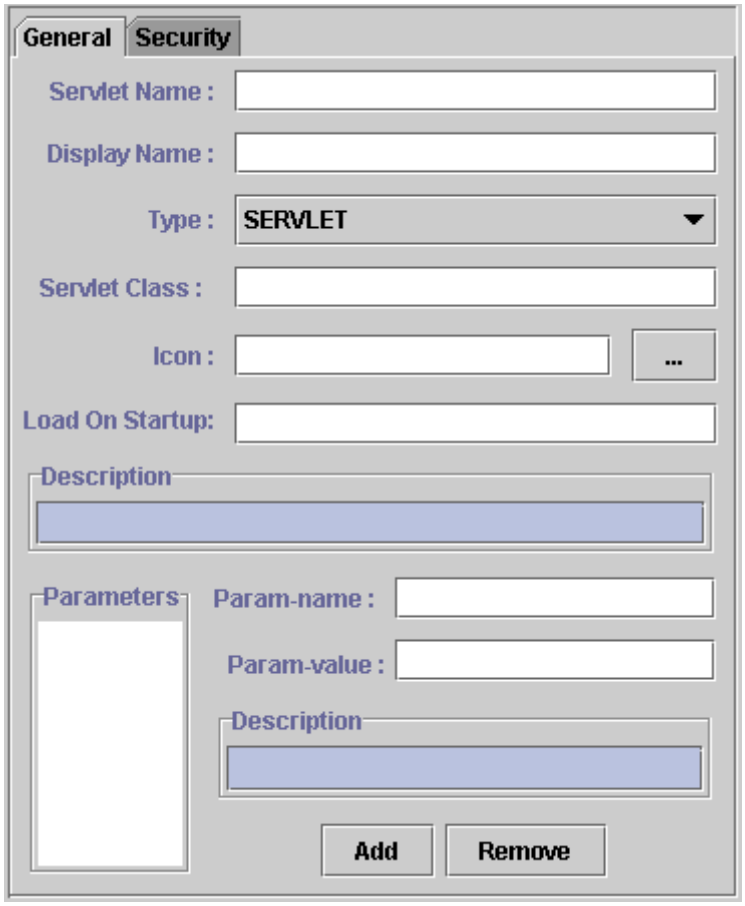
The basic properties of a Servlet or JSP can be edited using the two tabs in the right-hand pane – "General" and "Security."

General Tab

- "Servlet Name" – the name of the Servlet
- "Display Name" – contains a short name that is displayed by GUI tools
- "Type" – specifies the type of the file – `Servlet` or JSP
- "Servlet Class" or "JSP File Name" – the fully qualified Servlet class name or the full path to a JSP file within the Web application
- "Icon" – the image used to display the Web application in a GUI tool
- "Load on Startup" – indicates whether the Servlet is loaded at the start of the Web application. If no value is specified, or if the value specified is not a positive integer, the container is free to load it at any time in the startup sequence. If a JSP file is specified and `Load on Startup` value is given, then the JSP must be precompiled and loaded.
- "Description" – provides descriptive text about the Servlet
- "Parameters" pane – lists initialized parameters of the Servlet or JSP
- "Param-name" – specifies the parameter name
- "Param-value" – specifies the parameter value
- "Description" – displays a descriptive text about the initialization parameter

If a Servlet or JSP is added, and after the switch between the tabs is made, you must verify whether the Servlets or JSP class is correct. This verification consists of two steps. First, the class entered in "Servlet class" (or "JSP file" for JSP) must be added

as an additional file. Second, the mapping for the Servlet or JSP class must be correct. The mapping name for Servlet classes must begin with *WEB-INF/classes*. If one of the requirements is not met, an error message appears.

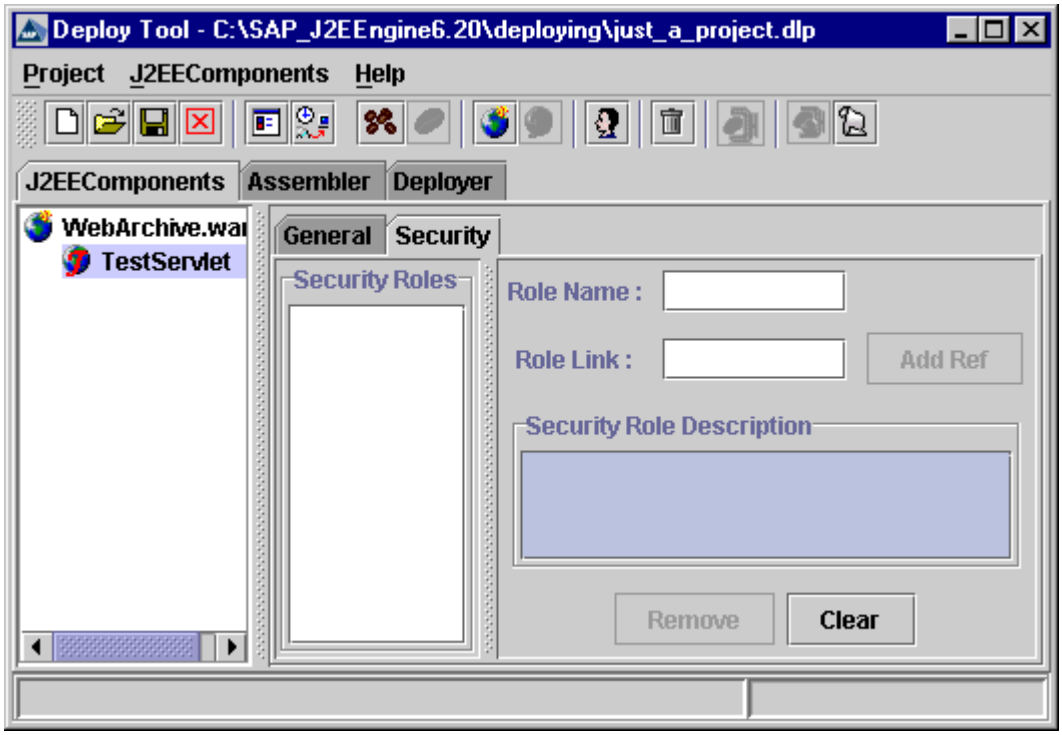


The Servlet's General Tab

Security Tab

One or more security roles can be defined in this tab using the following options:

- "Role Name" field – specifies the name of the security role to be added to the "Security Roles" pane
- "Role Link" field – specifies the name of the link to a defined security role, or the name of the security role reference. An existing security role can be added as a role reference. If this role does not exist, it must be created in the same tab in the "Assembler" tab. A security role can has only one security role reference.
- "Security Roles" pane – displays a list of available security roles and security role references
- "Security Role Description" – contains a descriptive text about the security role or security role reference



The “Security” Tab

Add Java Client Archive

To add Java Client Archives to the project, use the *J2EEComponents→Add Client* menu, or icon on the toolbar. Specify the Java Client Archive name in the dialog box that appears.

The “Add Client Jar” Window

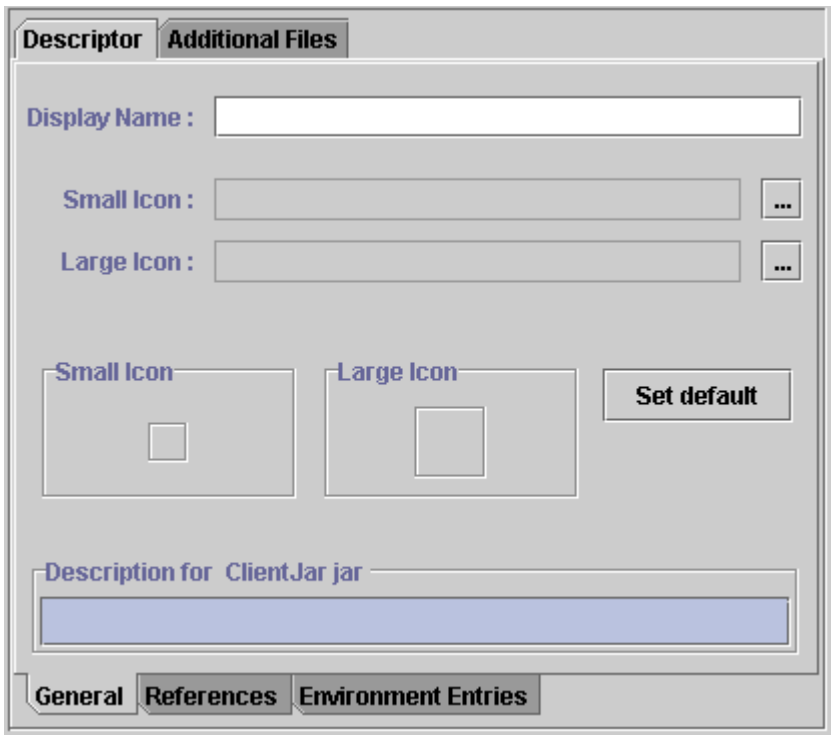
Select the Java Client Archive from the left-hand pane. Two tabs appear in the right-hand pane: “Descriptor” and “Additional.”

Descriptor Tab

This tab has three subtabs – “General,” “References,” and “Environments Entries.”

- “General” tab – provides the following information about the JAR file
  - “Display Name” – a short name is displayed by tools that use this JAR
  - “Small Icon” – specifies the full path to a small (16 x 16) icon image. When specified the icon appears in the “Small Icon” pane.
  - “Large Icon” – specifies the full path to a large (32 x 32) icon image. When specified the icon appears in the “Large Icon” pane.

- “Set default” – restores default icon settings
- “Description” – descriptive text about the selected JAR file

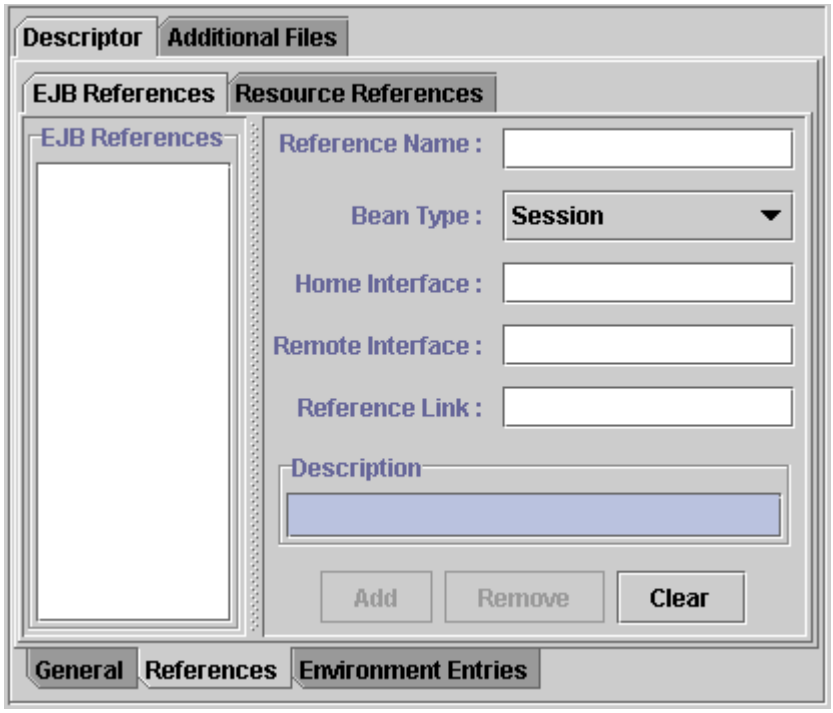


The “General” Tab

- “References” tab – provides options for linking the Web components to other application components or data sources. This tab contains two subtabs – “EJB References” and “Resource References.”

The “EJB References” subtab consists of the following fields:

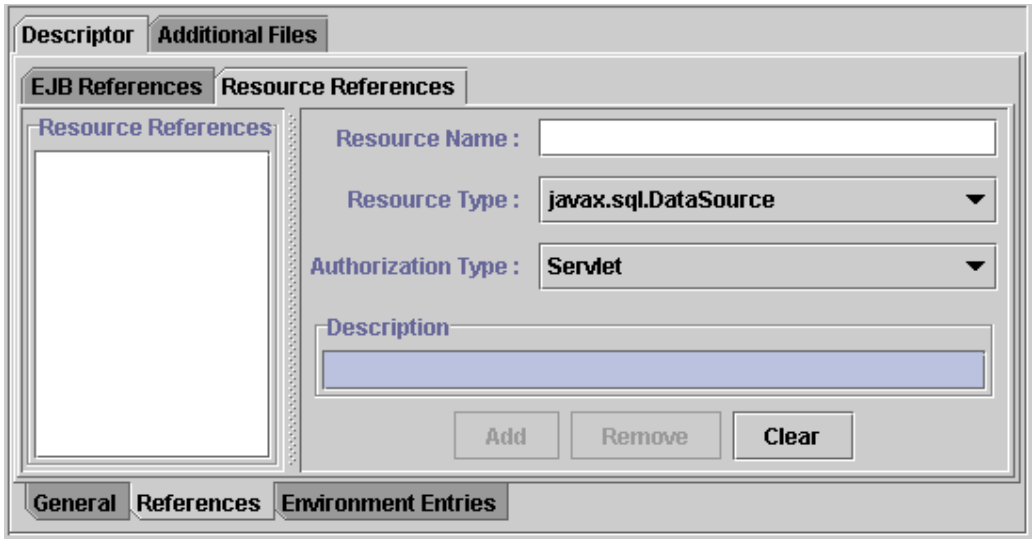
- “EJB References” pane – displays a list of all EJB’s references to other beans. The list contains the names of the referenced Beans.
- “Reference Name” – the name the EJB uses to refer to the referenced Bean
- “Bean Type” – specifies the type of the referenced Bean – `Entity` or `Session`
- “Home Interface” – specifies the fully qualified home interface name of the referenced Bean
- “Remote Interface” – specifies the fully qualified remote interface name of the referenced Bean
- “Reference Link” – specifies the lookup name of the referenced Bean
- “Description” – displays a short text about the reference



The “EJB References” Tab

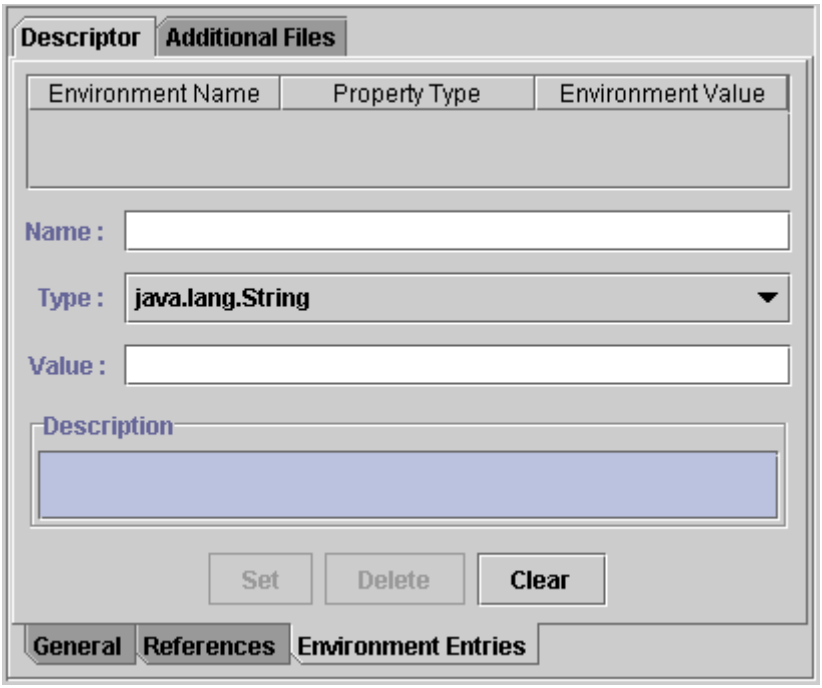
The “Resource References” subtab provides the following information:

- “Resource Name” – the name of the environment entry used in the EJB source code
- “Resource Type” – the type of the resource manager connection factory that the Bean code expects. It can be one of the following types `javax.sql.DataSource`, `javax.jms.QueueConnectionFactory`, `javax.jms.TopicConnectionFactory`, `javax.mail.Session`, `javax.resource.cci.ConnectionFactory` and `java.net.URL`.
- “Authorization Type” – specifies the responsibility for configuring the sign-on information of the resource manager. It can be either `Servlet` or `Container`.
- “Description” – displays a short text about the reference



The “Resource References” Tab

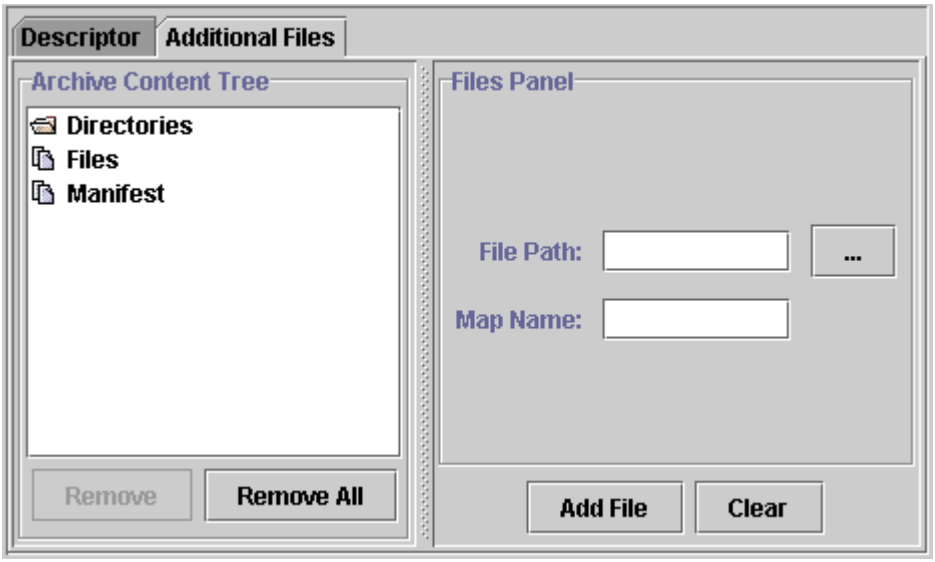
- “Environment” tab – you can customize the EJBs without accessing or changing their source code. This tab lists all specified environment entries with their “Environment Name,” “Property Type” and “Environment Value.” It provides the following information
  - “Name” – specifies the environment entry name
  - “Type” – specifies the fully qualified Java type of the environment entry value, expected by the EJB code. The possible values are `java.lang.Boolean`, `java.lang.String`, `java.lang.Integer`, `java.lang.Float` and `java.lang.Double`.
  - “Value” – specifies the current value of the property. It must be a valid string, according to the environment entry type.
  - “Description” – displays a short text of the environment entry



The “Environment Entries” Tab

Additional Tab

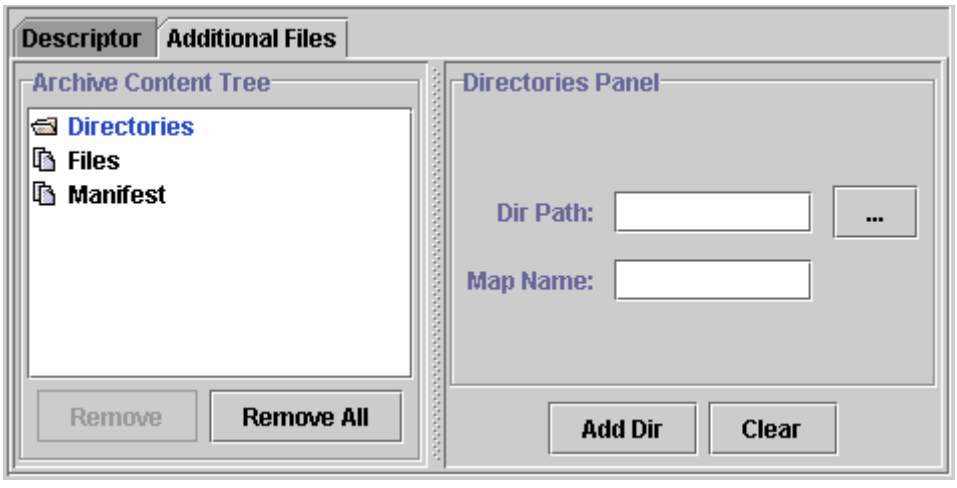
The “Additional” tab appends files or whole directories to a Web Archive file.



The “Additional Files” Tab

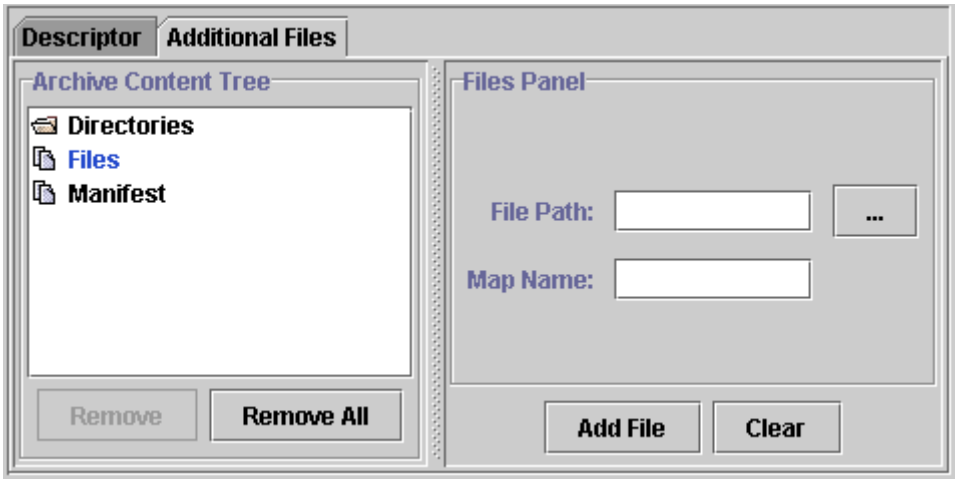
The tab contains the following options:

- “Archive Content Tree” pane – displays a list of directories, files, and the manifest file added to the archive
  - To add directories, select “Directories” from the “Archive Content Tree” pane. Select “Add Dir,” and a “Select Directory” dialog box appears. Browse to the desired directory and choose “Ok.” When a directory is appended, all its files and subdirectories are added to the WAR file. All files are specified by their relative paths. The added directories appear as tree nodes in the “Archive Content Tree” pane. Using each directory checkbox, you can choose which directories to add to the project.



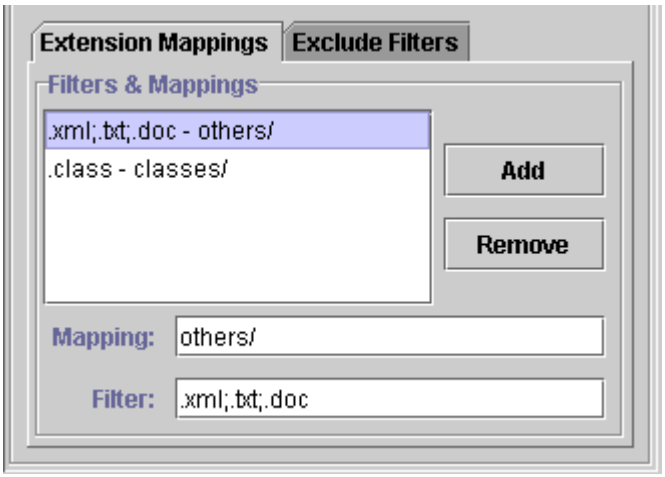
The “Directories Panel”

  - To add files, select “Files” from the “Archive Content Tree” pane. The “File Path” field in the “Files Panel” specifies the file path. The “Map Name” field specifies the name and type of the file to be appended to the JAR file. The “Add File” confirms the file was added. The files are added to the Client JAR file with their relative paths. The added files appear as tree nodes in the “Archive Content Tree” pane.



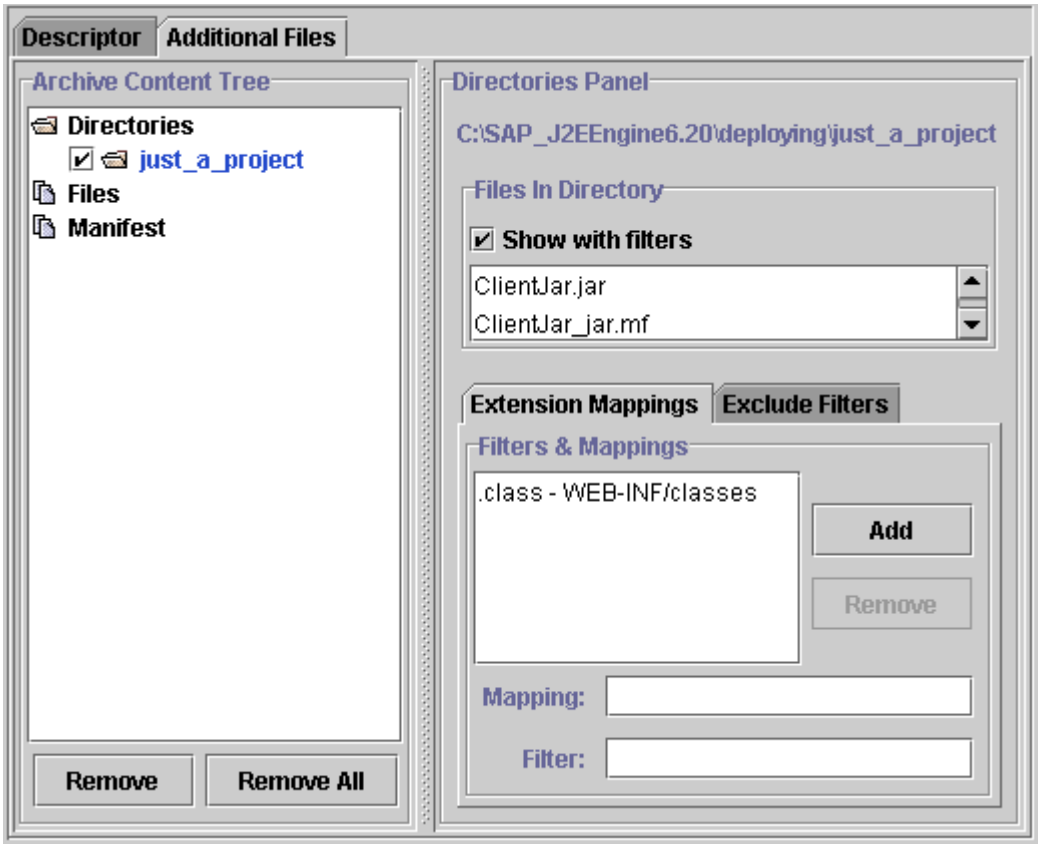
The “Files Panel”

- “Directories panel” is displayed when a directory is selected from the “Archive Content Tree” pane. The selected folder is displayed with its name, and the full path is displayed in “Directories Panel.” This panel contains the following features
  - “Files In Directory” – if the “Show with filters” box is set, the “Files In Directory” pane displays all files from the selected directory filtered with the specified filters. These files are added to the JAR file.
  - “Extension Mappings” tab – specifies the mapping of some types of files you want to add to the JAR file. This option helps to increase understanding and improve structure of the JAR file. The “Filters & Mappings” pane displays a list of all mappings and filters, defined for the selected directory using the “Mapping” and “Filter” fields. “Mapping,” specifies the directory where the filtered files are to be added. If the “Mapping” field points to an already added JAR file directory, the filtered files are added to the files in that directory. If the “Mapping” field points to a non-existent directory, a new directory with this name is created in the JAR file. The “Filter” field specifies the extension by which the files are filtered, and then added to the directory specified in the “Mapping” field. When specifying more than one filter, separate them using a semicolon. Use “Add” and “Remove” to change the list in the “Filters & Mappings” pane.



The “Extension Mappings” Tab

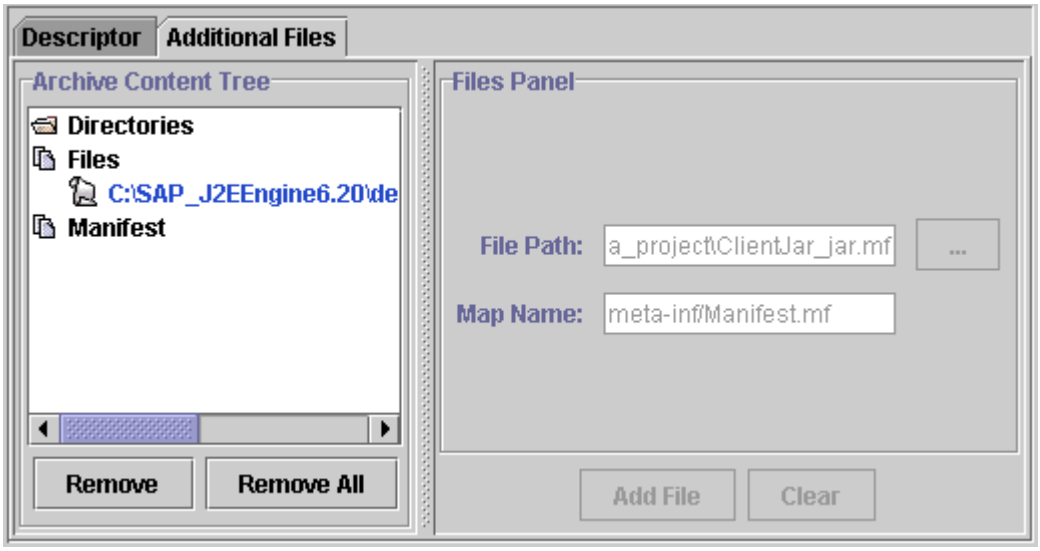
- “Exclude Filters” – displays a list of all filters for files that are to be excluded from the selected directory. When specifying more than one filter, separate them using a semicolon. Use “Add” and “Remove” to change the list in the “Exclude Filters” pane.



The “Directories Panel”

Note: Directories that are not filtered by include or exclude filters are added to the project with their original structure.

- “Files Panel” is displayed when the “Files” component is selected from the “Archive Content Tree” pane. This feature is useful to add a single file. It contains the following fields
  - “File Path” – specifies the path to the file to be added to the archive
  - “Map Name” – specifies the name that the file has in the archive



The “Files Panel”

- “Manifest Panel” – according to the J2EE™ Specification, all Java archive files have a manifest file. The “Manifest Panel” enables you to edit the manifest file. For the Java Client Archive you must set information from type: `MAIN-CLASS: <ClassName>` for the main class to be started directly from the Java Client Archive. When a manifest file is added, it appears as a tree node in the “Files” tree. The Manifest file path is relative. The “Map Name” field value is *META-INF/MANIFEST.MF*.

Remove

Choose *J2EEComponents→Remove*, or on the toolbar, to remove a selected component from the “J2EEComponents” tab.

Make Archives

To make an archive file choose *J2EEComponents→Make Archive*, or on the toolbar. An EAR file appears in the “Components” tab for the selected components only. This file is created automatically with the name of the project file.

Make All Archives

To make archive files for all J2EE components, choose *J2EEComponents→Make All Archives,* or icon on the toolbar.

View Log File

To view the event log file choose *J2EEComponents→View Log File,* or on the toolbar. You must make the EAR archive file before viewing the event log file. Otherwise, a message warns that there is no log file.

Application Assembling

The “J2EEComponents” tab is used to make archive files – EJB JARs, Web archives (WARs), ApplicationClient JARs – in accordance with J2EE, EJB 1.1, and Servlet 2.2 Specifications. All archive files must be assembled in an EAR file to be deployed

The “Assembler” tab combines different archives (JARs and WARs) in an application that is generated as an EAR file. The archives can be created either using the “J2EEComponents” tab, or by another IDE.

The “Assembler” tab provides the following options:

Add Archive

To add a JAR or WAR file to the J2EE EAR, choose *Assemble→Add Archive,* or  on the toolbar. You only have to enter the file path. This option is useful when there is an existing archive.

Remove

To remove a selected archive file from the J2EE EAR, choose *Assemble→Remove,* or  on the toolbar. This command removes only archive files.

Edit the Archive Descriptors

In the “Assembler” tab, you can edit the environment entries’ values, and modify the references to the EJBs in the same application. You can also synchronize the security role names and define additional roles. In this tab, you must specify the transaction attributes, if they are not specified. The access to methods must also be specified in accordance with the application security preferences.

All archive properties that are changed in the “Assembler” tab are saved in an alternative descriptor file. The deployment descriptor file created in the“J2EEComponents” tab has the name *<archivename_extension_DD>*, and the alternative descriptor file in the “Assembler” tab is named: *<archivename_extension_AltDD>.*

These two deployment descriptor files are merged each time the “Assembler” tab is accessed, or when the *Assemble→Refresh* command is executed. The following set of rules is used.

JAR Descriptor

- Merge Security roles – takes the defined Security roles from the JAR deployment descriptor. The new roles are taken from the alternative deployment descriptor.
- Merge Method permission – takes method permissions from the alternative deployment descriptor. It checks whether the same methods exist in the JAR’s remote and home interfaces. If method permissions for them are set, these permissions are set in the result deployment descriptor.

- Merge container transaction – gets transaction attributes from the alternative deployment descriptor. If transaction attributes for the same methods in the remote and home interfaces exist in the JAR deployment descriptor, these attributes are set in the result deployment descriptor.
- Merge EnvironmentEntry – sets the JAR deployment descriptor's environment entries in the result deployment descriptor. If there are environment entries in the alternative deployment descriptor that have the same names as the entries in the result deployment descriptor their values are taken from the alternative deployment descriptor and are set in the result deployment descriptor.
- Merge ResourceReference and EJBReference – gets references from the JAR deployment descriptor and sets them in the result deployment descriptor. If there are references with the same name in the alternative deployment descriptor, their reference links are set in the result deployment descriptor.

WAR Descriptor

- Merge Servlets descriptors – takes the defined Servlet deployment descriptors from the WAR file
- Merge Security roles – takes the defined Security roles from the WAR deployment descriptor. The new roles are taken from the alternative deployment descriptor.
- Merge ResourceReference and EJBReference – gets references from the WAR deployment descriptor and sets them in the result deployment descriptor. If there are references that have the same name in the alternative deployment descriptor, their reference links are set in the result deployment descriptor.
- Merge EnvironmentEntry – sets the WAR deployment descriptor's environment entries in the result deployment descriptor. If there are environment entries in the alternative deployment descriptor that have the same names as the entries in the result deployment descriptor their values are then taken from the alternative deployment descriptor and are set in the result deployment descriptor.
- Merge JSP Tag Libraries – gets the WAR deployment descriptor's JSP Tag Libraries and sets them in the result deployment descriptor. If there are JSP Tag Libraries with different Tag URI names in the alternative deployment descriptor, they are set in the result deployment descriptor.
- Merge Welcome Pages – takes WAR deployment descriptor's Welcome Pages and sets them in the result deployment descriptor. If there are Welcome Pages that have different names in the alternative deployment descriptor, they are set in the result deployment descriptor.
- Security constraints – the Web resource collection element is used to identify a subset of the resources and HTTP methods on those resources within a Web application to which a security constraint applies. If no HTTP methods are specified the security constraint applies to all HTTP methods.

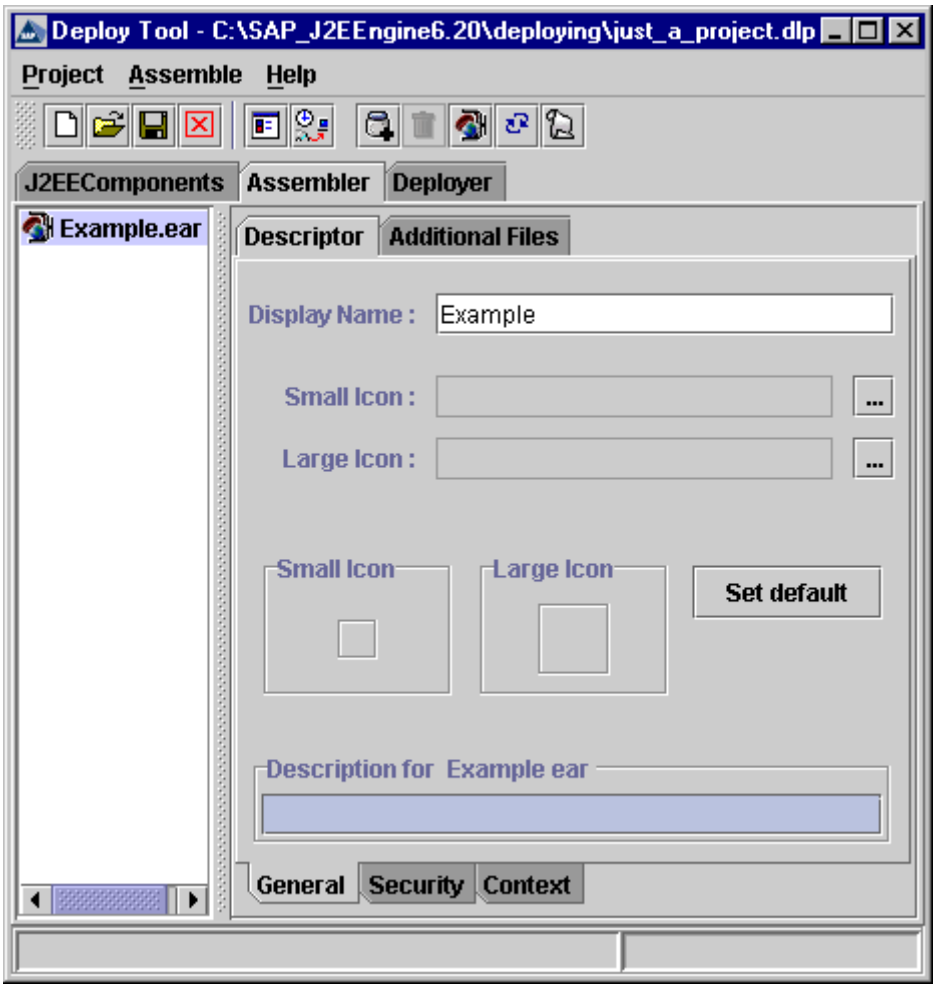
Client's JAR Descriptor

- Merge EnvironmentEntry – sets the client descriptor's environment entries in the result deployment descriptor. If there are environment entries in alternative deployment descriptor that have the same names as the entries in the result deployment descriptor, then their values are taken from the alternative deployment descriptor and are set in the result deployment descriptor.
- Merge ResourceReference and EJBReference – gets the References from the client deployment descriptor and sets them in the result deployment descriptor. If there are references with the same name in the alternative

deployment descriptor, their reference links are set in the result deployment descriptor.

Edit the J2EE EAR

Select the EAR file. The right-hand pane displays two subtabs – “Descriptor” and “Additional Files.”



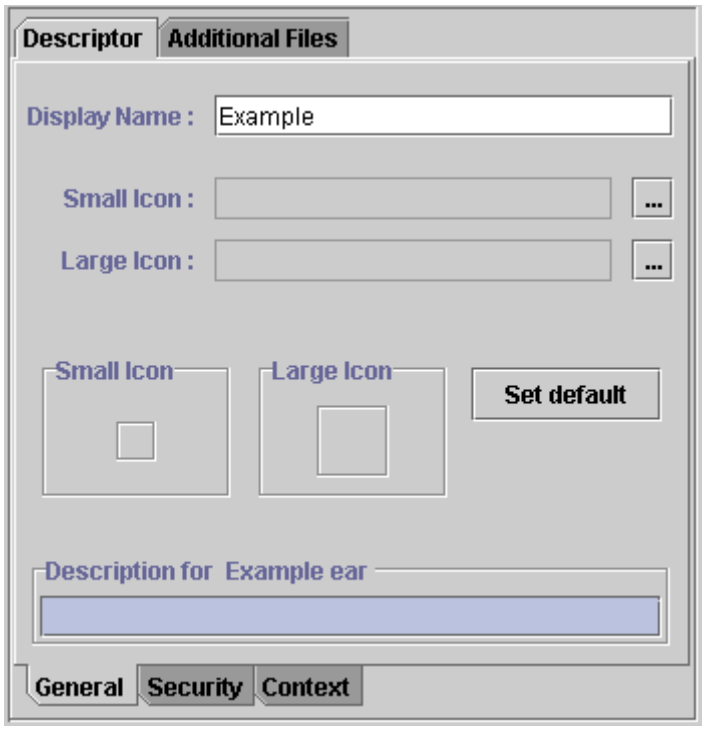
The “Descriptor” Tab

Descriptor Tab

The “Descriptor” tab has three subtabs – “General,” “Security,” and “Context.”

- “General” tab – provides the following information about the EAR file
  - “Display Name” – a short name that is displayed by tools that use this EAR
    Note: In the “Display Name”, do not use the following special symbols: slash (/), double backslash (\\), double quotation (“), colon (:), asterisk (*), question mark (?), backslash (\), greater than sign (>), less than sign (<) and vertical slash (|). They will be replaced with underscore (_). Do not end the name with period (.).
  - “Small Icon” – specifies the full path to a small (16 x 16) icon image. When specified, the icon appears in the "Small Icon" pane.

- “Large Icon” – specifies the full path to a large (32 x 32) icon image. When specified, the icon appears in the "Large Icon" pane.
- “Set default” – restores the default icon settings
- “Description” – a descriptive text about the selected EAR file



The “General” Tab

- “Security” tab – one or more security roles can be defined in this tab using the following options
  - “Role Name” field – specifies the name of the security role to be added to the Security Roles list
  - “Role Link” field – specifies the name of the link to a defined security role, or the name of the security role reference. An existing security role can be added as a role reference. If this role does not exist, it must be created in the tab with the same name in the “Assembler” tab. A Security role can have only one security role reference.
  - “Security Roles” list – displays a list of available security roles and security role references. The “Security Role Description” textbox – contains a descriptive text about the security role or security role reference.

The “Security” Tab

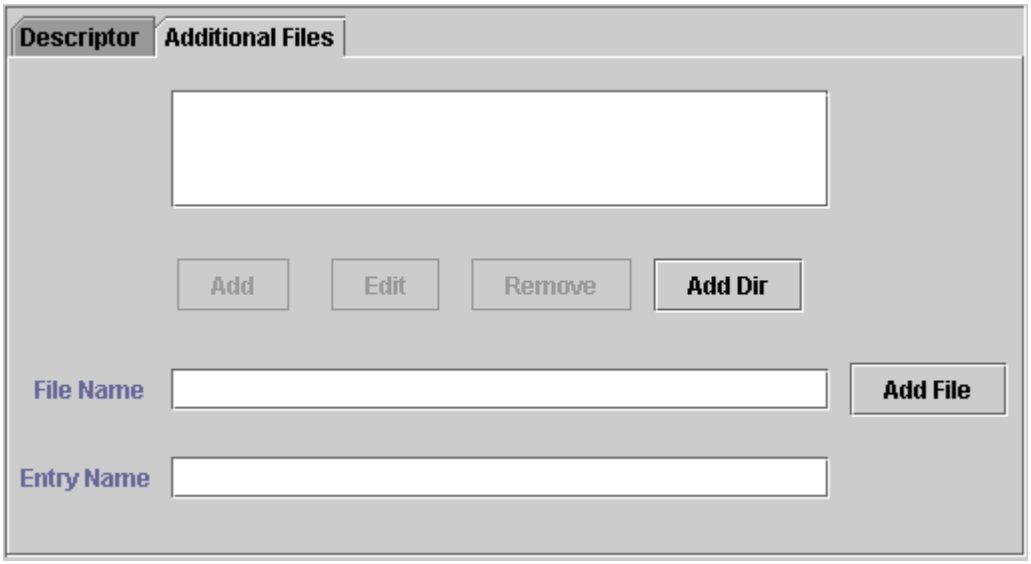
- “Context” tab – specifies the WAR context root under which the WAR is accessible from the HTTP Service

The “Context” Tab

Additional Files Tab

The “Additional Files” tab appends files to an EAR file.

- “Add Dir” – appends all files from a selected directory and its subdirectories to the EAR file. All files are displayed in the “Additional Files” pane.
- “Remove” – removes selected file from the EAR file
- “Edit” – edits the “Entry Name” of a file
- “Add File” or “...” browses to a file
- “Add” – add selected file



The “Additional Files” Tab

Additional Options

The “Assembler” tab provides some additional options:

- *Assemble→Refresh,* or on the toolbar – refreshes the project when the archives are changed from an external IDE, or when the tabs are changed.
- *Assemble→Make EAR*, or on the toolbar – makes the EAR file. The EAR must have a unique name. If two applications have identical names after deployment, errors may occur.
- *Assemble→View Log File*, or on the toolbar – displays the service log file. You must make the EAR before viewing the service log file. Otherwise, a message warns that there is no log file.

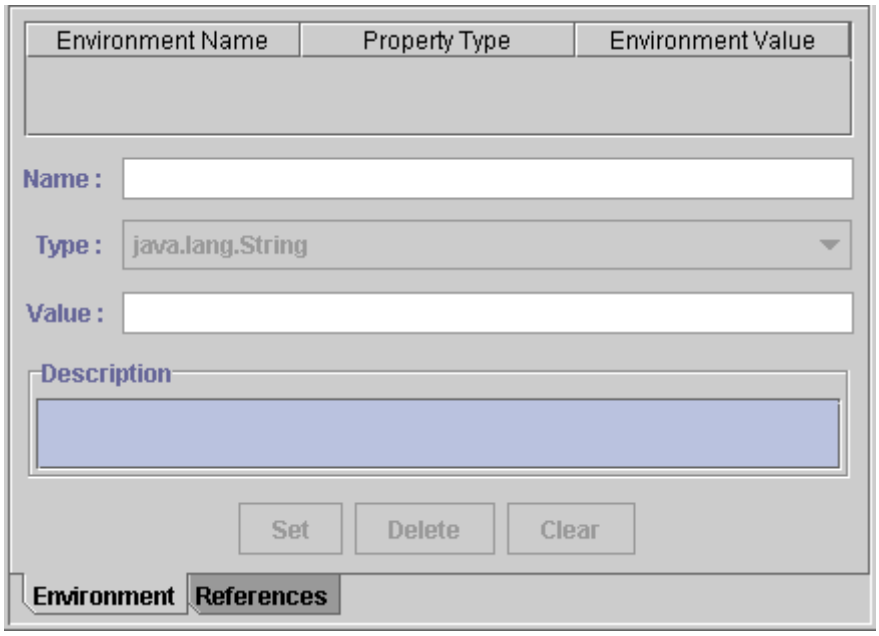
Deployment

To deploy your application, use the SAP J2EE Engine Deploy Tool “Deployer” tab. This enables you to you to modify the J2EE Engine dependent properties. The storage properties must be mapped to the available databases. The security role mappings can be modified only after the Deploy Tool is connected to the J2EE Engine. When you establish connection between Deploy Tool and SAP J2EE Engine, you can map the defined security roles to real users and you can modify the security role references. You can also specify a new role link reference, but you cannot remove the existing role references. In the “Deployer” tab, you can add a Resource Link, User name, and Password for a WAR file, EJB, or client JAR, only if the Resource References are already set in the “J2EEComponents” or “Assembler” tab. In this tab, you can also edit the properties value, but you cannot set additional properties.

Note: In Deploy Tool in the *Deployer* → *Environment* tab, after you have modified a property value of an existing entry in a table or a list, choose “Set” or “Add” to confirm the changes before saving the project file. Otherwise, the changes are not stored.

Note: To deploy or re-deploy an application correctly, make sure that you have closed all services and *Work* subdirectories on the server side, and that no other application is permitted to read or write at the time of deployment. Otherwise, a “Compiling Exception” occurs when starting the application. This is a platform-dependant problem. When there are open Explorers under Windows in the directory, Java programs cannot write files and they receive no error message. This causes the program to assume that files are written, but they are actually missing.

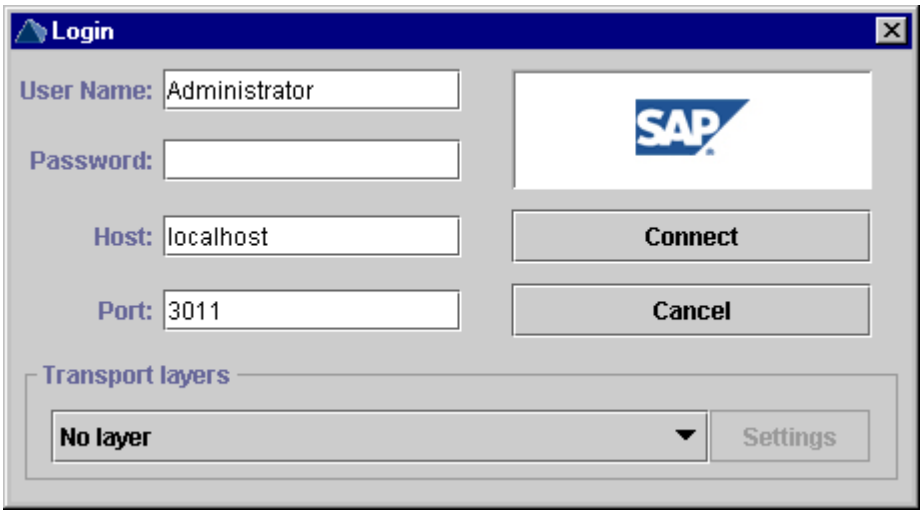
Note: Do not restart EJB or Web Container while deploying, updating, undeploying, starting or stopping application.



The “Environment” Tab

The “Deployer” tab provides the following properties:

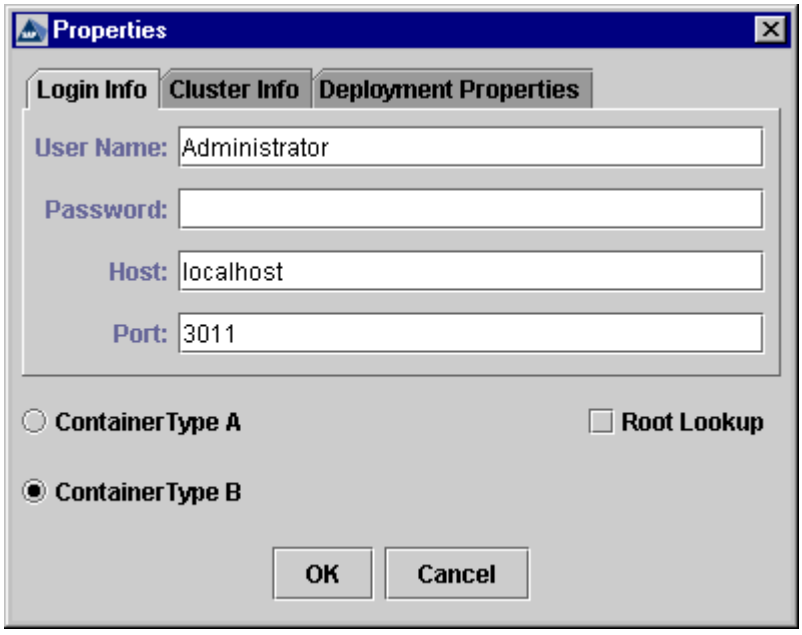
- *Deploy→EAR→Load Ear,* or on the toolbar – loads the EAR. This command is efficient when a created EAR is being deployed, or when the “Assembler” tab has not been used.
- *Deploy→EAR→Unload Ear* – removes the EAR file from the “Deployer” tab
- *Deploy→EAR→Refresh,* or on the toolbar – refreshes the project when archives are changed from an external IDE, or when some of the tabs are changed
- *Deploy→Connect/Disconnect,* or / icon on the toolbar – if the login information is set correctly connects or disconnects the user from the J2EE Engine. Login information can be set in the “Login” window.



The “Login” Window

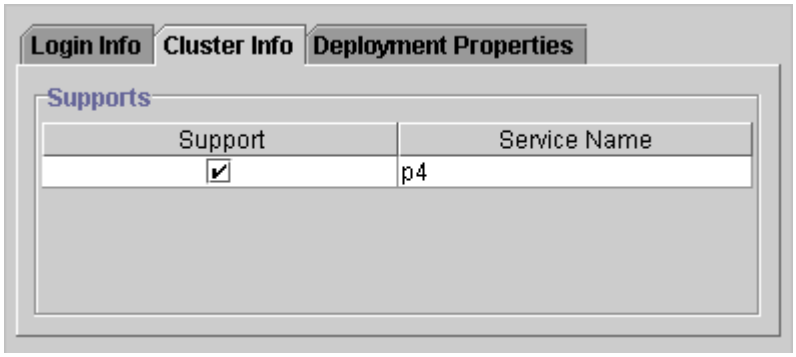
- *Deploy→Deployment→Deploy Ear,* or on the toolbar – deploys the EAR on SAP J2EE Engine
- *Deploy→Deployment→Update* – redeploys an existing application. If there are additional components, they are added. The application remains the same, even if some errors occur during the update. A quicker way to update a changed application is to undeploy the old one and to deploy the new. This command is efficient for applications that do not have a lot of corrections in their components.
- *Deploy→Deployment→Undeploy* – removes the application from the J2EE Engine. If you start the command after a connection between the Deploy Tool and the cluster has been established, the Deploy Tool displays a table of previously deployed applications and enables you to select the ones you want to undeploy. This is useful when more than one application must be removed. If you start the command without being connected to the cluster, you must specify the correct application name.
- *Deploy→ Properties* – this command opens a dialog box where you can modify the cluster properties. Use “Container Type” to specify the container type – type A or B. Type A uses a single Bean instance for parallel transactions that access one Primary Key. Type B uses different Bean instances for one Primary Key for each transaction. “Root Lookup” checkbox is also available. It must be set for applications that have to lookup from the context root, but not from their own subcontext. Three tabs are available in the “Cluster Properties” dialog box – “Login Info,” “Cluster Info” and “Deployment Properties.”

- “Login Info” – enables you to modify the user name, password, host, and port settings



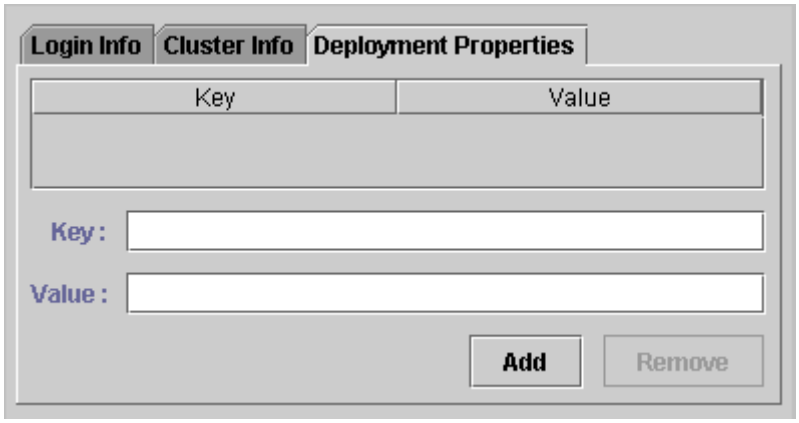
The “Login Info” Tab

- “Cluster Info” – enables you to modify the cluster information used for the deployment process. Choose the protocol the clients use to connect to the application.



The “Cluster Info” Tab

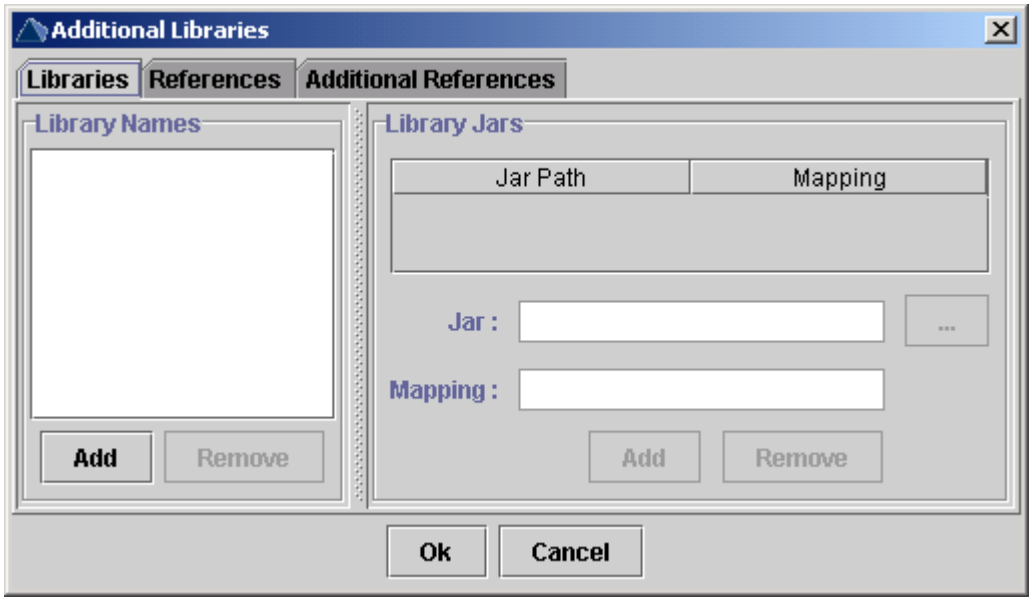
- “Deployment Properties” – specifies additional properties and their values, for the deployment process



The “Deployment Properties” Tab

- *Deploy→Libraries* – adds additional libraries and references to the application. The following subtabs are available:
  - “Libraries” – adds an external library to be used from the application. Each library must have at least one JAR file. All additional libraries to be deployed, along with the application are displayed in a list in the “Library Names” pane. They can be added or removed from it. When the library is added to an application, a reference to the library is created. When it is removed, the reference can be kept or removed. This feature is useful for applications that use archive files from already deployed libraries. Additional JAR files can be added to a library with a specified JAR path and Mapping name. All JAR files that belong to a library are displayed in a list in the “Library Jars” pane. The “Mapping” field specifies the file path used to add the JAR files to the library.

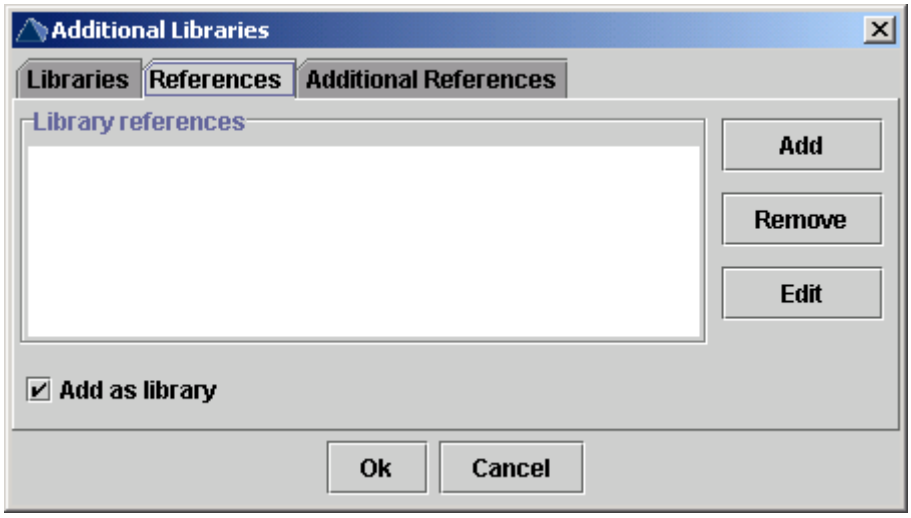
Note: Each library must have at least one library file.



The “Libraries” Tab

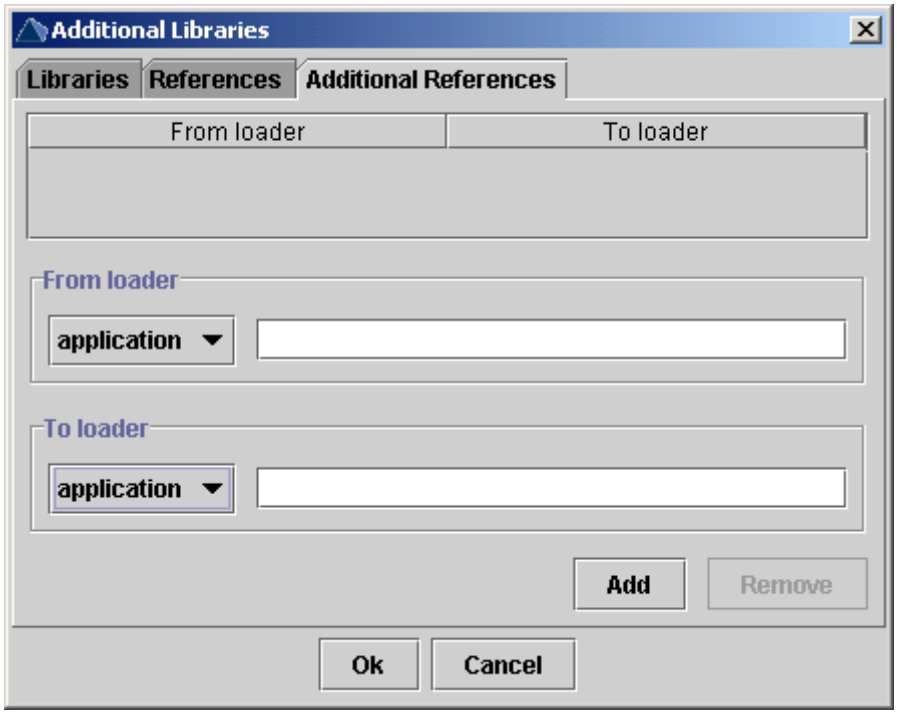
  - “References” – specifies references from the application to the libraries, application and/or services. All references are displayed in list in the “Library reference” pane. They can be added or removed from it. The

"Add as library" checkbox specifies whether the reference from the application is to a library, or to another application.



The "References" Tab

- "Additional References" – specifies additional references from application, service or library to another application, service or library. Define the name of the application, service or library that uses the reference as "From loader." Define the name of the application, service or library that is used from the reference as "To loader." It is obligatory each "From loader" to have at lest one "To loader."



The "Additional References" tab

- *Deploy→View Log File,* or on the toolbar – displays the event log file

Storage.xml

Every JAR that contains entity EJB must have a *Storage.xml* file that specifies the storage properties of the entity Bean. The Deploy Tool generates this file when you use Deployer tab. The XML must have the following data type description:

```
<!DOCTYPE EntityEJBStorages [
<!ELEMENT EntityEJBStorages (EntityEJB)*>
<!ELEMENT EntityEJB (Storage)>
<!ATTLIST EntityEJB Name CDATA #REQUIRED>
<!ELEMENT Storage (RDBStorageProps)>
<!ATTLIST Storage StorageName CDATA #REQUIRED ClassName CDATA
#REQUIRED>
<!ELEMENT RDBStorageProps (FieldColumnMap*, rdbFinderDescriptor*)>
<!ATTLIST RDBStorageProps PoolName CDATA #REQUIRED TableName CDATA
#REQUIRED>
<!ELEMENT FieldColumnMap (#PCDATA)>
<!ATTLIST FieldColumnMap Field CDATA #REQUIRED Table CDATA
#REQUIRED Column CDATA #REQUIRED Key CDATA #REQUIRED>
<!ELEMENT rdbFinderDescriptor (MethodArgument*)>
<!ATTLIST rdbFinderDescriptor MethodName CDATA #REQUIRED Criteria
CDATA #REQUIRED >
<!ELEMENT MethodArgument (#PCDATA)>
]>
```

All fields are explained below:

Note: The `DOCTYPE` field must be specified properly for the XML file to be parsed correctly.

```
<!--
EntityEJBStorage element is the root element.
This element is obligatory and the name has to be
EntityEJBStorage.
-->
<!ELEMENT EntityEJBStorages (EntityEJB)*>

<!--
EntityEJB element defines entity EJB that has storage properties.
Used in: EntityEJBStorages
-->
<!ELEMENT EntityEJB (Storage)>

<!--
Name attribute specifies the Bean name of an entity Bean.
If no Name is specified an empty string is put as a default value
in the XML.
Used in: EntityEJB
-->
<!ATTLIST EntityEJB Name CDATA #REQUIRED>

<!--
Storage element specifies all RDBStorageProps properties.
Used in: EntityEJB
-->
<!ELEMENT Storage (RDBStorageProps)>

<!--
StorageName and ClassName attribute specifies the storage type and
the class that implements it. Currently, the only available
storage type is RDB Storage and the available ClassName is
com.inqmy.services.ejb.deploy.descriptors.dbstorage.RDBMSStoragePr
operties.
```

```
Used in: Storage
-->
<!ATTLIST Storage StorageName CDATA #REQUIRED ClassName CDATA
#REQUIRED>

<!--
RDBStorageProps element specifies RDBStorageProps by pool and
table name.
Used in: Storage
-->
<!ELEMENT RDBStorageProps (FieldColumnMap*, rdbFinderDescriptor*,
Joins*)>

<!--
PoolName and TableName attributes specify the pool name and table
name of the used database. To deploy the application properly be
sure that Pool and Table with specified names exist at the SAP
J2EE Engine.
Used in: RDBStorageProps
-->
<!ATTLIST RDBStorageProps PoolName CDATA #REQUIRED TableName CDATA
#REQUIRED>

<!--
FieldColumnMap element defines the correspondence between the
table fields of the database and the EJB's fields.
Used in: RDBStoragePrpos
-->
<!ELEMENT FieldColumnMap (#PCDATA)>

<!--
Field, Table, Column and Key attributes describes the field in the
entity EJB. For example they explain where to map the bean. Key
specifies if this field is a primary key or a regular table
column. The possible values are "1" and "0" respectively.
Used in: FieldColumnMap
-->
<!ATTLIST FieldColumnMap Field CDATA #REQUIRED Table CDATA
#REQUIRED Column CDATA #REQUIRED Key CDATA #REQUIRED>

<!--
The rdbFinderDescriptor element describes the finder methods of
the entity beans.
Used in: RDBStoragePrpos
-->
<!ELEMENT rdbFinderDescriptor (MethodArgument*)>

<!--
The MethodName and Criteria attributes describe the bean's finder
method and search criteria.

The MethodName specifies the name of the method.

The Criteria describes the SQL statement of the finder method and
contains the WHERE clause of the SELECT statement. If you use more
complicated SELECT statements, you can describe the whole SELECT
statement in the criteria element. Use $ (the dollar sign) and a
number to indicate the parameters of the finder method in the SQL
statement.
Example:
If the finder method in the bean's source code is:
findYoungEmployees(String name, int age)
the Criteria element can contain either of the following SQL
statements:
age_column<$2 and name_column=$1
or
```

```
select * from EmployeesTable where age_column <$2 and name_column
=$1

It is not obligatory to describe a Criteria for the
findByPrimaryKey method. If the Criteria is empty, all table
fields are selected.

Used in: rdbFinderDescriptor
-->
<!ATTLIST rdbFinderDescriptor MethodName CDATA #REQUIRED Criteria
CDATA #REQUIRED>

<!--
The MethodArgument element defines the argument types of the
finder method. For the example above – java.lang.String and int.
Used in: rdbFinderDescriptor
-->
<!ELEMENT MethodArgument (#PCDATA)>
```

Example:

```
  <EntityEJBStorages>
    <EntityEJB Name="IS/Bug">
      <Storage ClassName="com.ejb.RDBMSStorageProperties"
StorageName="RDB Storage">
        <RDBStorageProps PoolName="ISPool" TableName="IS_BUG">
          <FieldColumnMap column="IS_BUG_bugId" Field="bugId"
Key="1" Table="IS_BUG">
          </FieldColumnMap>
          <FieldColumnMap column="IS_BUG_bugStatusId"
Field="bugStatusId" Key="0" Table="IS_BUG">
          </FieldColumnMap>
          <FieldColumnMap column="IS_BUG_className"
Field="className" Key="0" Table="IS_BUG">
          </FieldColumnMap>
          <FieldColumnMap column="IS_BUG_creationDate"
Field="creationDate" Key="0" Table="IS_BUG">
          </FieldColumnMap>
          <FieldColumnMap column="IS_BUG_description"
Field="description" Key="0" Table="IS_BUG">
          </FieldColumnMap>
          <rdbFinderDescriptor Criteria="" MethodName="findAll">
          </rdbFinderDescriptor>
          <rdbFinderDescriptor Criteria="IS_BUG_bugStatusId = $1"
MethodName="findByBugStatusId">
            <MethodArgument>
              int
            </MethodArgument>
          </rdbFinderDescriptor>
          <rdbFinderDescriptor Criteria="IS_BUG_creationDate = $1"
MethodName="findByCreationDate">
            <MethodArgument>
              java.sql.Date
            </MethodArgument>
          </rdbFinderDescriptor>
          <rdbFinderDescriptor Criteria="IS_BUG_description like
'%' + $1+ '%'" MethodName="findByDescription">
            <MethodArgument>
              java.lang.String
            </MethodArgument>
          </rdbFinderDescriptor>
          <rdbFinderDescriptor Criteria="IS_BUG_productId = $1"
MethodName="findByProductId">
            <MethodArgument>
```

```
              int
            </MethodArgument>
          </rdbFinderDescriptor>
        </RDBStorageProps>
      </Storage>
    </EntityEJB>
    <EntityEJB Name="IS/BugStatus">
      <Storage ClassName="com.ejb.RDBMSStorageProperties"
StorageName="RDB Storage">
        <RDBStorageProps PoolName="ISPool"
TableName="IS_BUGSTATUS">
          <FieldColumnMap column="IS_BUGSTATUS_bugStatusId"
Field="bugStatusId" Key="1" Table="IS_BUGSTATUS">
          </FieldColumnMap>
          <FieldColumnMap column="IS_BUGSTATUS_status"
Field="status" Key="0" Table="IS_BUGSTATUS">
          </FieldColumnMap>
          <rdbFinderDescriptor Criteria="" MethodName="findAll">
          </rdbFinderDescriptor>
          <rdbFinderDescriptor Criteria="IS_BUGSTATUS_status like
$1" MethodName="findByStatus">
            <MethodArgument>
              java.lang.String
            </MethodArgument>
          </rdbFinderDescriptor>
        </RDBStorageProps>
      </Storage>
    </EntityEJB>
  </EntityEJBStorages>
```

Chapter 3

Getting Started with Deploy Tool

- Example on using Deploy Tool

Introduction

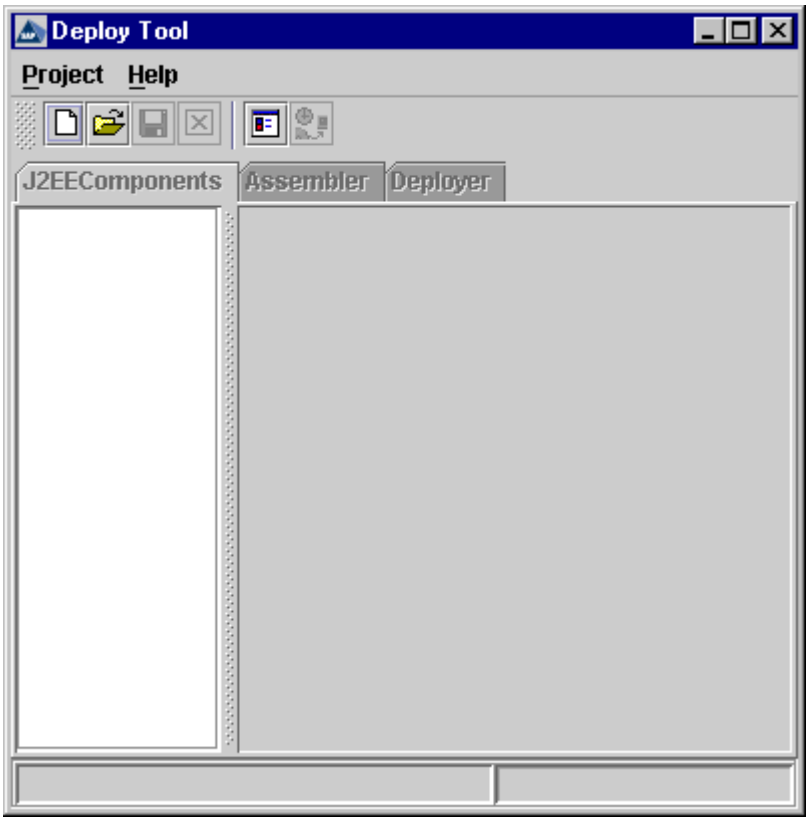
Deploy Tool is a graphic user interface that provides the easiest way to deploy applications on SAP J2EE Engine. Deploy Tool must be installed along with the installation of SAP J2EE Engine. There is an application example in *<SAPj2eeEngine_install_dir>/deploying/deploy_example*. This example represents an HTML page that calls a Servlet with two parameters. The Servlet parses them to integer values and gives these parameters to a stateful session Bean. The Bean multiplies the values and returns the result. All necessary source files are located in *deploy_example* directory. The HTML page is the *index.html* file in *<SAPj2eeEngine_install_dir>/deploying/deploy_example*. The Bean interface files are in *<SAPj2eeEngine_install_dir>/deploying/ deploy_example/test1*. The Servlet class file is in *<SAPj2eeEngine_install_dir>/deploying/deploy_example/WEB-INF/classes/test1*. There are additional files in these folders that are necessary for the direct deployment of the application example, without using the Deploy Tool. For example, executing the *example* script file, located in *<SAPj2eeEngine_install_dir>/deploying/deploy_example* deploys the application example. The additional files located in *<SAPj2eeEngine_install_dir>/deploying/deploy_example* are the already assembled components and deployment descriptors needed for the deployment.

This tutorial is to explain how to assemble the required application components and to deploy the resulting EAR on the J2EE Engine. This tutorial does not use existing archives, XML deployment descriptors and script file. Application components are assembled in a separate project directory. The deployment procedure for this example is described below.

Step 1: Start the Deploy Tool

To start Deploy Tool execute *DeployTool* script file located in *<SAPj2eeEngine_install_dir>/deploying*.

The Deploy Tool initial frame appears:



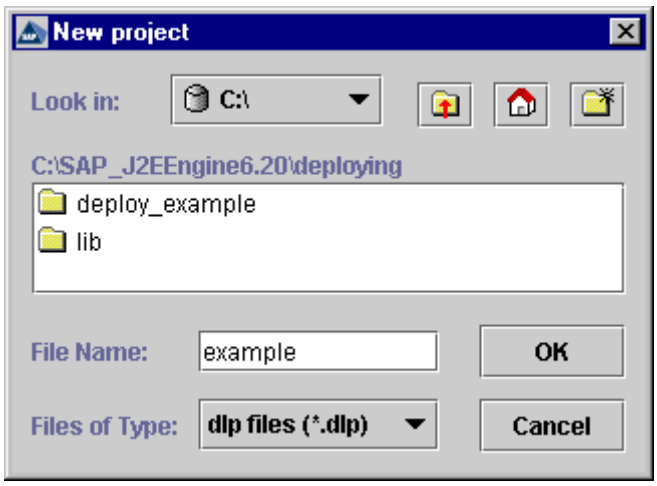
The Deploy Tool Initial Frame

The Deploy Tool is run, and you can now create a deploy project file. The deploy project file is a file with *dlp* extension containing all the necessary information about the application you deploy.

Step 2: Create a New Project File

Choose *Project→New Project*, or icon on the toolbar. The “New project” window appears. Specify a valid project filename of your choice. A filename cannot contain any of the following character: slash (/), backslash (\), colon (:), asterisk (*), question mark (?), quotation mark (“), greather than sign (<), less than sign (>) or vertical slash (|). The project file is saved in *<SAPj2eeEngine_install_dir>/deploying* directory, and a subdirectory with the project’s name is created containing all the necessary files for the deployment of your application.

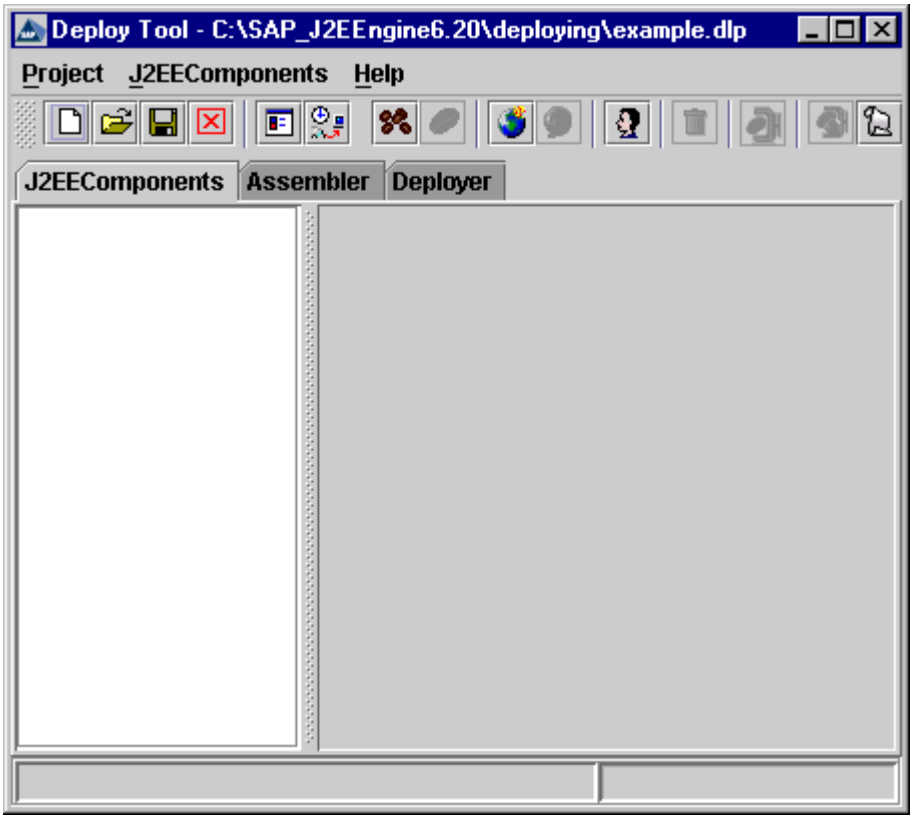
Name the example project *example.dlp*. Enter “example” in the “File Name” of the “New Project” dialog, and choose “OK.”



The New Project

The empty project file named *example.dlp* appears in the Deploy Tool “Main” frame.

J2EEComponents appears at the main frame with some additional icons at the toolbar.



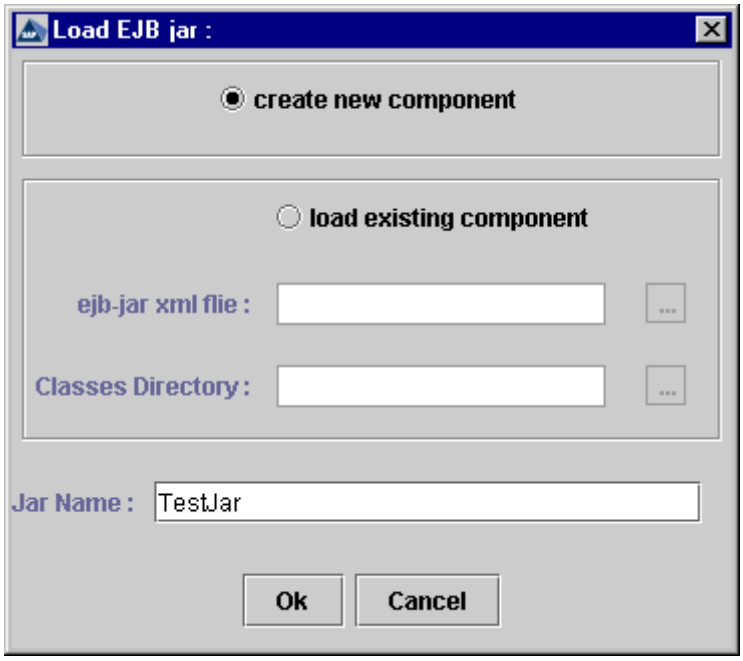
The Empty Project File – *example.dlp*

You can add EJB Group, Web Archive or Java Client Archive to the project. EJBs must be put in an EJB Group (*JAR* archive file), Web components such as JSP and Servlets must be added to a Web Archive group (*WAR* archive file), and Java clients must be put into a client archive group (*JAR* archive file). These JARs and WARs are assembled according to the J2EE™ Specification.

The example contains an EJB and a Servlet class. Therefore, an EJB Group (for the EJB) and a Web Archive (for the Servlet class) must be added.

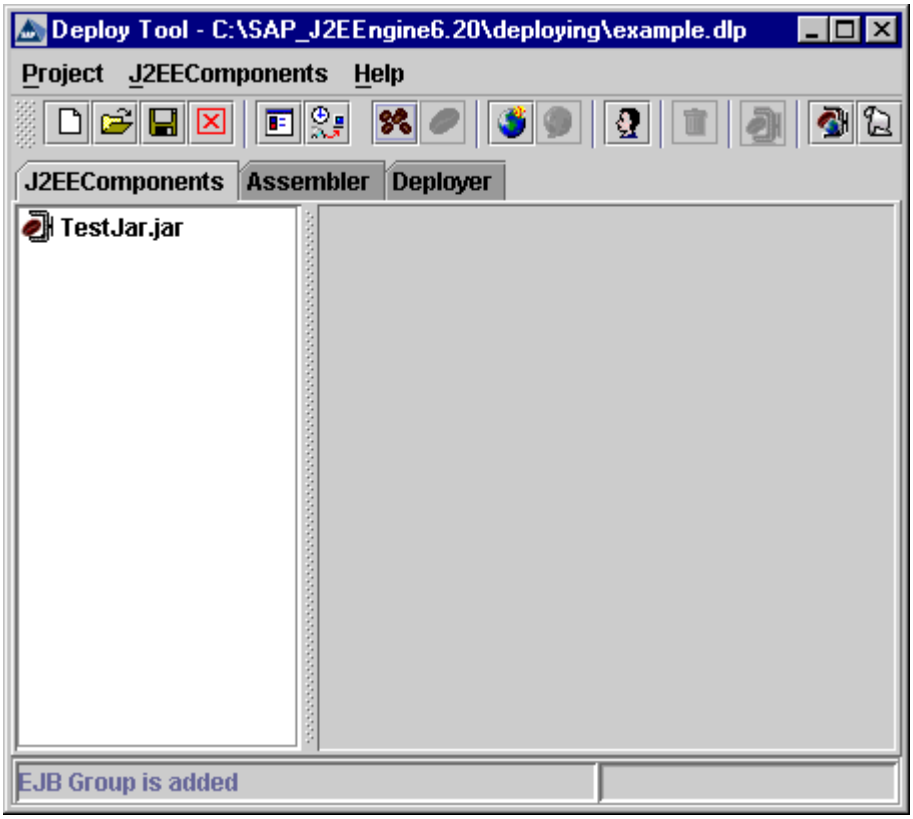
Step 3: Add an EJB Group

Choose *J2EE Components→Add EJB Group*, or  on the toolbar. The “Load EJB JAR” window appears.



Load EJB JAR Window

Set the “Create New Component” and enter “TestJar” in the “Jar Name” field. Choose “Ok.” The EJB Group is added to the project. The Deploy Tool has the following view:

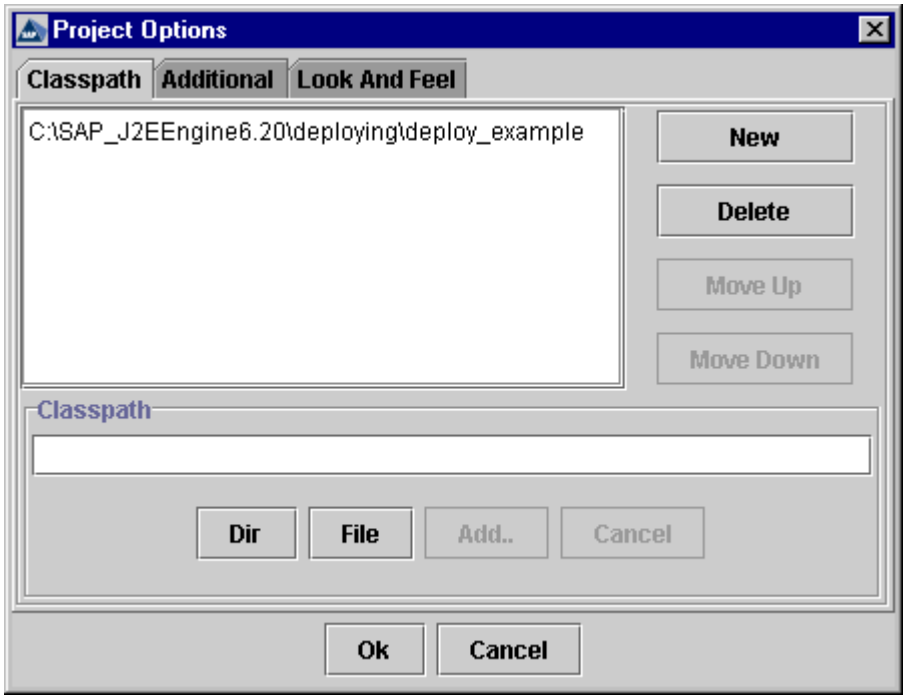


The New EJB Group

Step 4: Set the Project Classpath

Each EJB Group has some Java files. The example contains EJB and Servlet classes. Therefore, their classpaths must be specified. Before adding the components, specify the classpaths.

If you set incorrect classpaths and the compiler cannot find the specified classes, an error message box appears. Choose *Project→Options...* or  icon on the toolbar to set the necessary classpaths. Select the “Classpath” tab. Choose “New.” Select “Dir” and browse to select the example directory, as shown below:



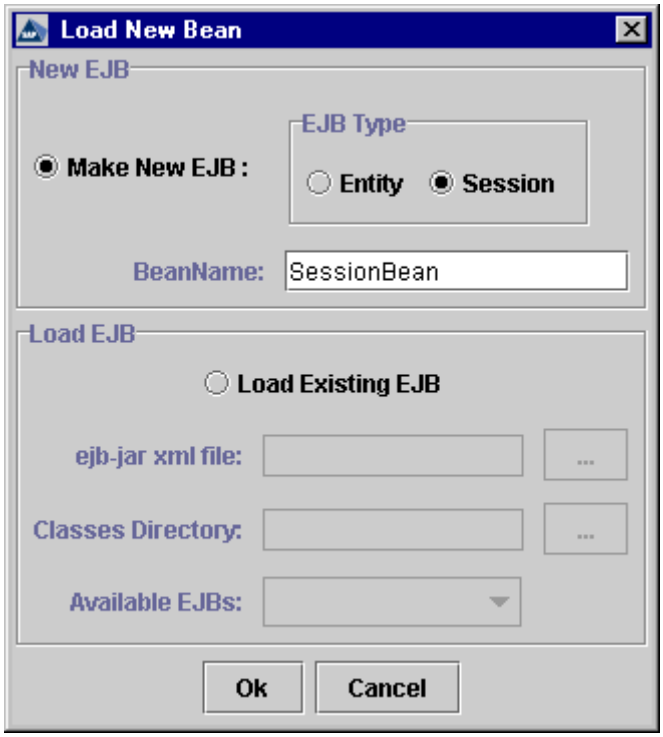
Specify the Project Options

Choose “Add...” to add the selected classpath to the list. The “Add...” is active only when a new classpath is typed, or when a new directory or file classpath is chosen. Choose “Ok” to save the changes and to close the “Project Options” dialog box.

Note: If, for example, the EJB’s class files are not in the same directory (although they are in equivalent packages), you must set the classpath to each of these sources (directories, archive, and so on.). After returning to the “J2EEComponents” tab, switch between the “General” and any other subtab to verify that all classes have been found.

Step 5: Add the EJB

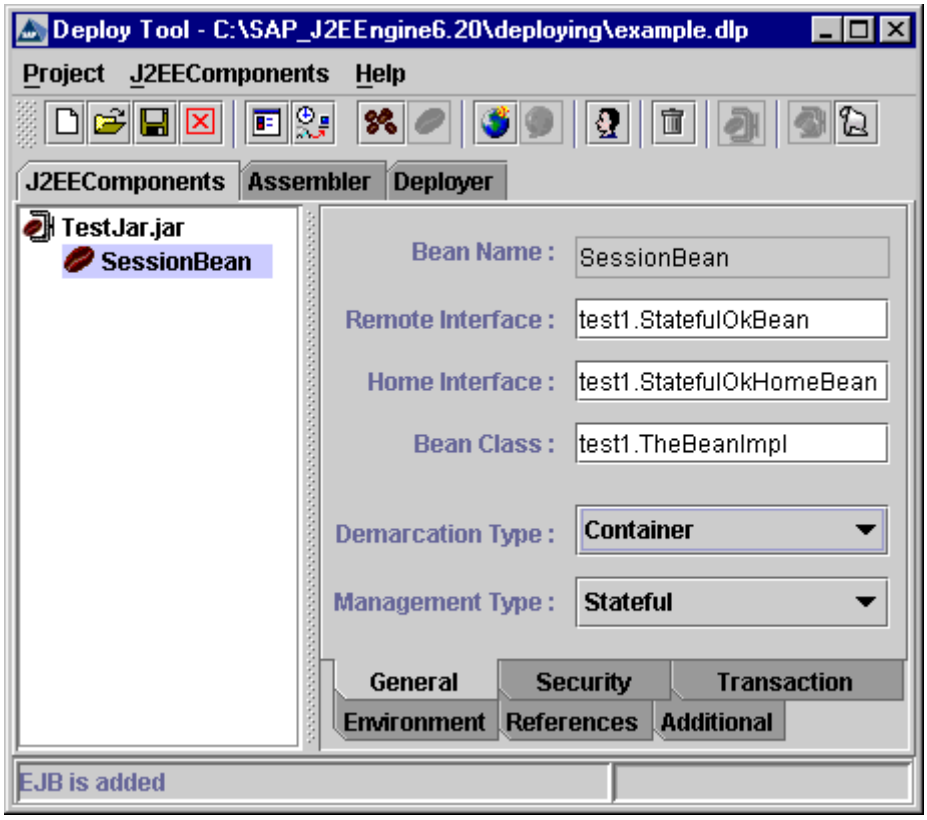
The example contains a session EJB. Therefore, the session EJB must be added to the EJB Group. Select the EJB Group the session EJB is joining. For this example, select "TestJar." Choose *J2EEComponents→Add EJB,* or on the toolbar. Enter all necessary information in the "Load New Bean" dialog box, as shown below:



Make New EJB

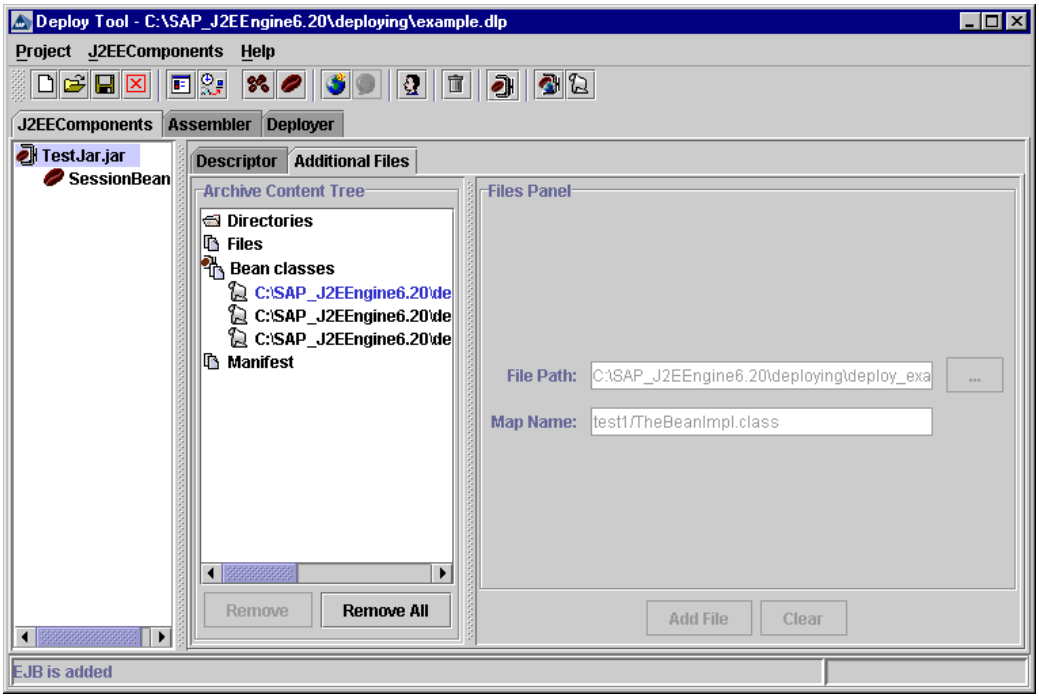
Choose "Ok." The new session EJB is created. It appears as a subnode of the EJB Group in the "J2EEComponents" tab.

To insert the EJB properly, its Remote Interface, Home Interface and Bean Class must be specified correctly in the corresponding fields. These names consist of the fully qualified Java class names. You can check the Bean's package and interface or class names in the source code files located in the *<SAPj2eeEngine_install_dir>/deploying/deploy_example/test1* folder. The "Demarcation Type" and "Management Type" fields are specific for session Beans. "Demarcation Type" defines the responsibilities for managing the transactions – Bean or Container. "Management Type" specifies the SessionBean type – Stateful or Stateless. The example's properties are shown below:



The EJB General Properties

The Bean's interfaces and "Project Classpath" must be set properly. If there is a problem an error message appears. If the EJB is set correctly, the "Additional Files" tab for the EJB Group shows all Bean classes.



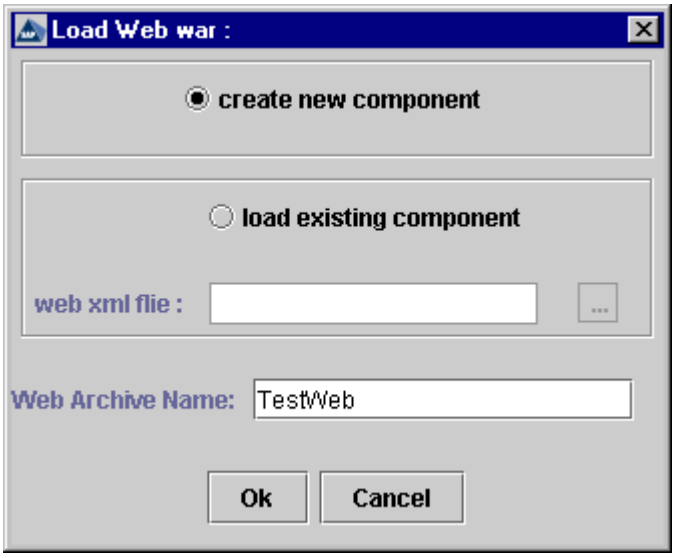
EJB Group "Additional Files" Tab

Deploy Tool generates *TestJar_jar.mf* file and adds it to the project with *META-INF/MANIFEST.MF* mapping name.

Note: Do not delete the TestJar_jar.mf manually. Manifest file is important for the application deployment.

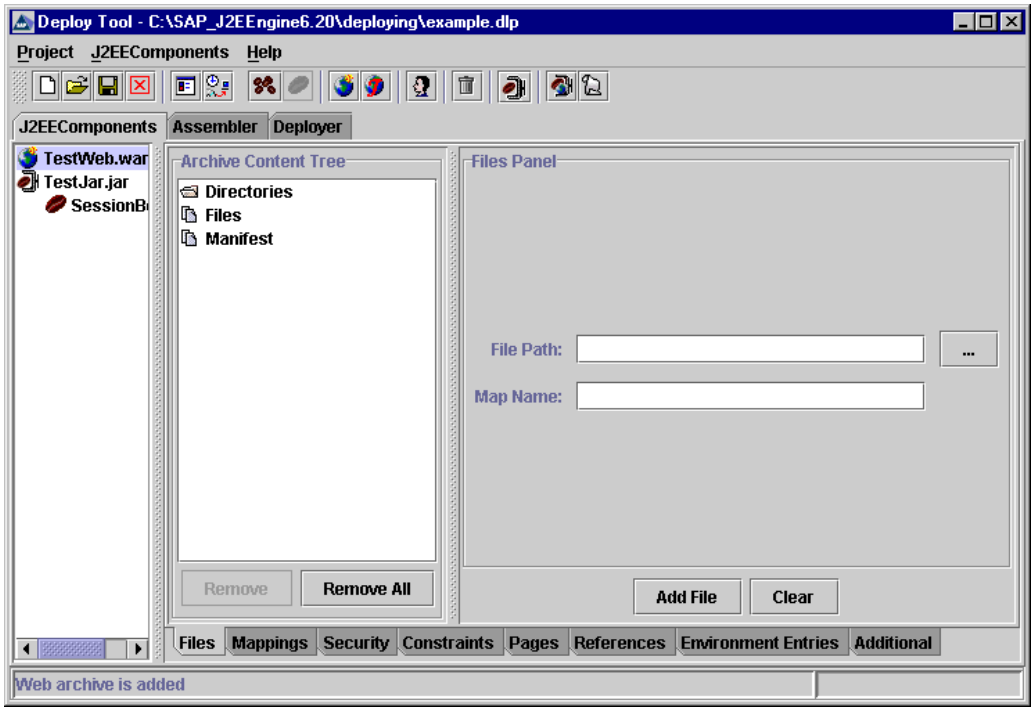
Step 6: Add a WEB Archive Group

Choose *J2EEComponents→Add Web*, or  on the toolbar to create a new Web group. The “Load Web WAR” dialog box appears. Select “create new component” and enter “TestWeb” as a Web archive name in the corresponding field. This field is required. The “Load Web war” dialog box is shown below:



“Load Web WAR” Dialog Box

Choose “Ok.” The new “TestWeb” Web Archive group appears in the left-hand frame of “J2EEComponents” tab:



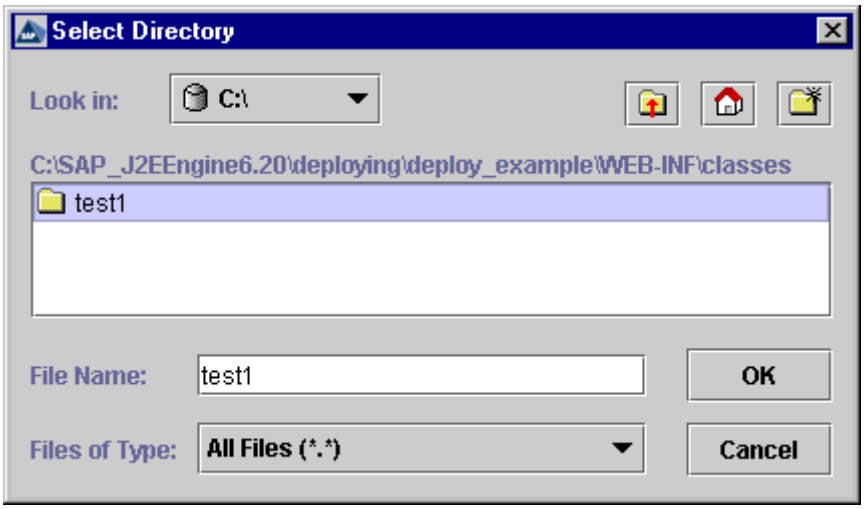
The TestWeb Web Archive Group

The next step is to add the Servlet to the Web archive. You can add a Servlet or JSP from file, or create a new one.

Step 7: Add a Directory to the WEB Archive

Entire directories with their files can be added to the Web Archive using the “Files” tab. Add a directory with the Servlet class to the example project.

Select Directories from the “Archive Content Tree” pane. The “Directories Panel” appears in the right-hand pane of the “Files” tab. Choose “...” and the “Select Directory” dialog box appears:



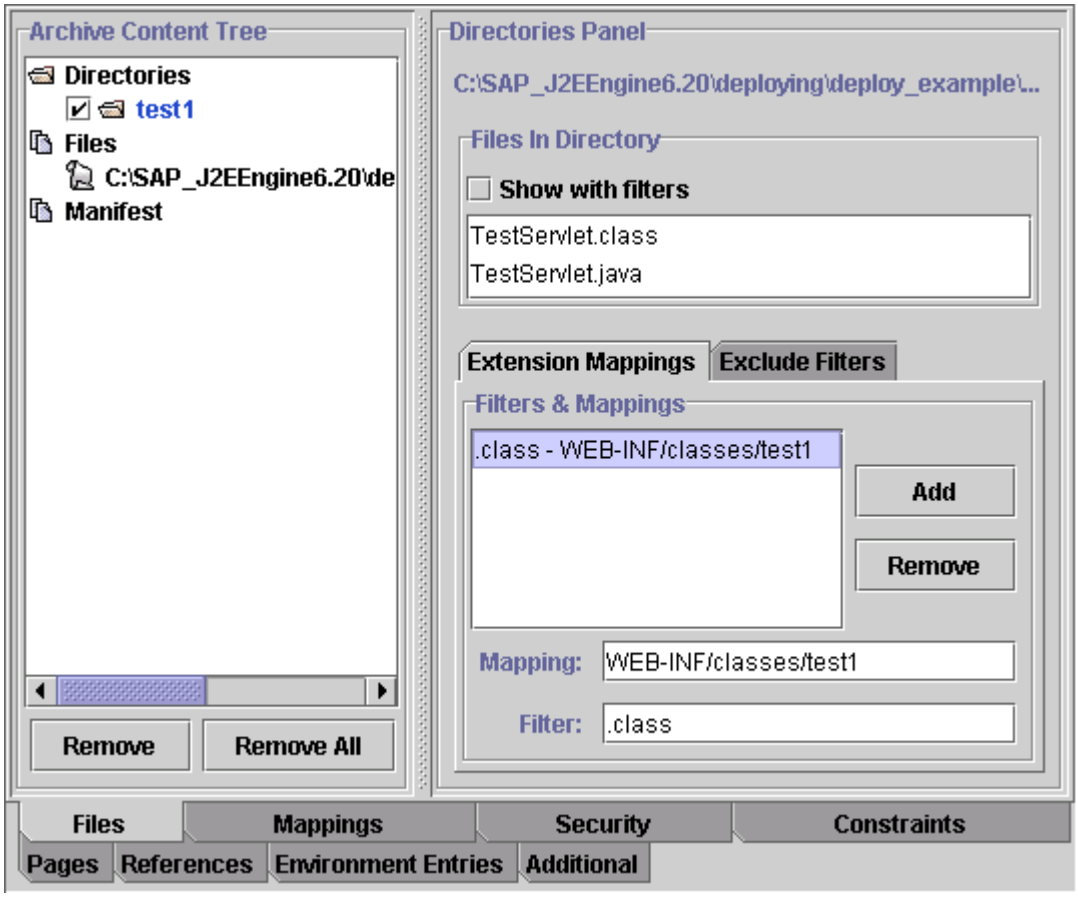
“Select Directory” Dialog Box

Browse to choose the location of Servlet class directory – *<SAPj2eeEngine_install_dir>/deploying/deploy_example/WEB-INF/classes/test1*. Select *test1* directory. Choose “Ok.”

The directory mapping can be used to change the file structure of an existing WAR. This can be done for better understanding and file structuring. You can change the directory files mapping later by selecting the directory in the “Archive Content Tree” pane and modifying its properties.

To add the example servlet to the WAR, set the mapping name “WEB-INF/classes/test1” to class files of the added directory..

The directory appears as a subnode of Directories in the “Archive Content Tree” pane.



Add a Directory to the WEB Archive

Step 8: Add Files to the WEB Archive

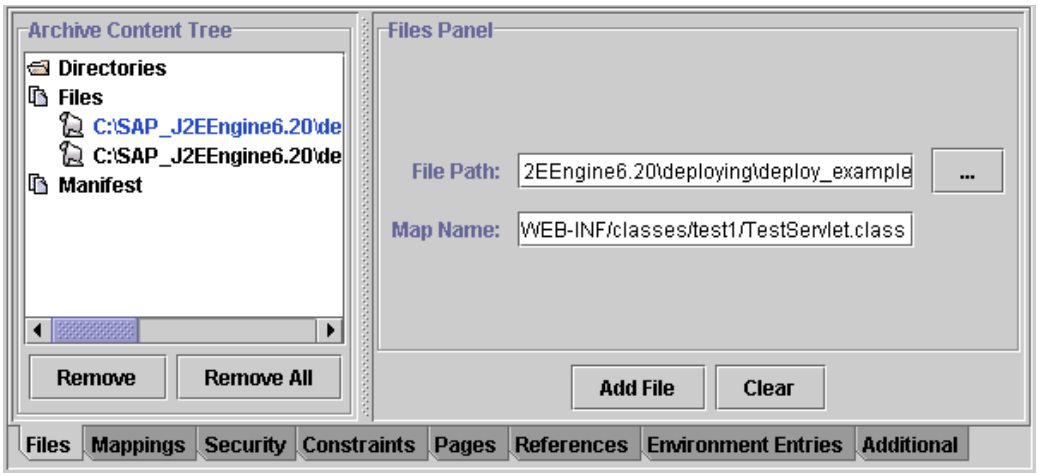
Note: You can add a Servlet or HTML file, using the instructions in *Step 7* (Add a Directory to the WEB Archive) or in *Step 8* (Add Files to the WEB Archive). If you use both steps, an error for duplication appears.

The example considered contains two files – an *html* file and a Servlet class file. Both files must be added to the Web Archive. The Servlet class must be added to the archive with the specified mapping.

Select the Web Archive node from the tree in "J2EEComponents" tab left-hand pane. The right-hand pane consists of several subtabs. Choose the "Files" subtab and select "Files" from the "Archive Content Tree" pane. The "Files Panel" pane contains two fields – "File Path" and "Map Name." To specify the absolute path to the Servlet class use the "File Path" field. To browse to a Servlet class use "..." on the right-hand side pane. The "Map Name" field specifies the relative path of the Servlet class within the WAR archive, and can be used to change the file location within the Web archive. If this field is left empty, the file is saved in the root directory of the application. If your project uses Applets, their class files must be saved in the project's root directory. If your project contains Servlets, they must be mapped to the *WEB-INF/classes/* directory.

The value of the "File Path" field for the Servlet class in the example is *<SAPj2eeEngine_install_dir>/deploying/deploy_example/WEB-INF/classes/test1/TestServlet.class*. The value of the "Map Name" field must be *WEB-INF/classes/test1/TestServlet.class*. Choose "Add File."

Repeat the same steps for the *html* file that invokes the Servlet. The value of the "File Path" field must be *<SAPj2eeEngine_install_dir>/deploying/deploy_example/index.html* and the value of the "Map Name" field must be *index.html*. Choose "Add File."



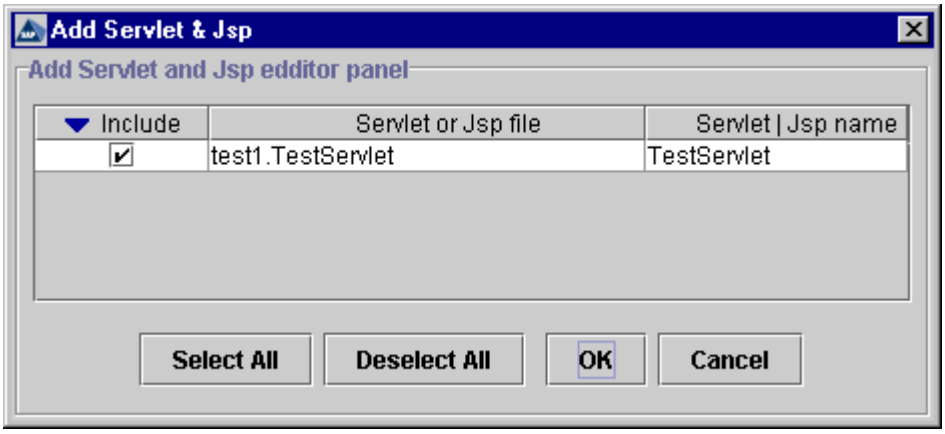
Add Files

Step 9: Add the Servlet

Select *TestWeb.war*. You can add a Servlet or JSP from file, or create a new one.

In the example project, a Servlet is added from the existing class file. The directory containing the Servlet class was added in the *Step 7*, and the mapping of the Servlet was specified in the previous step.

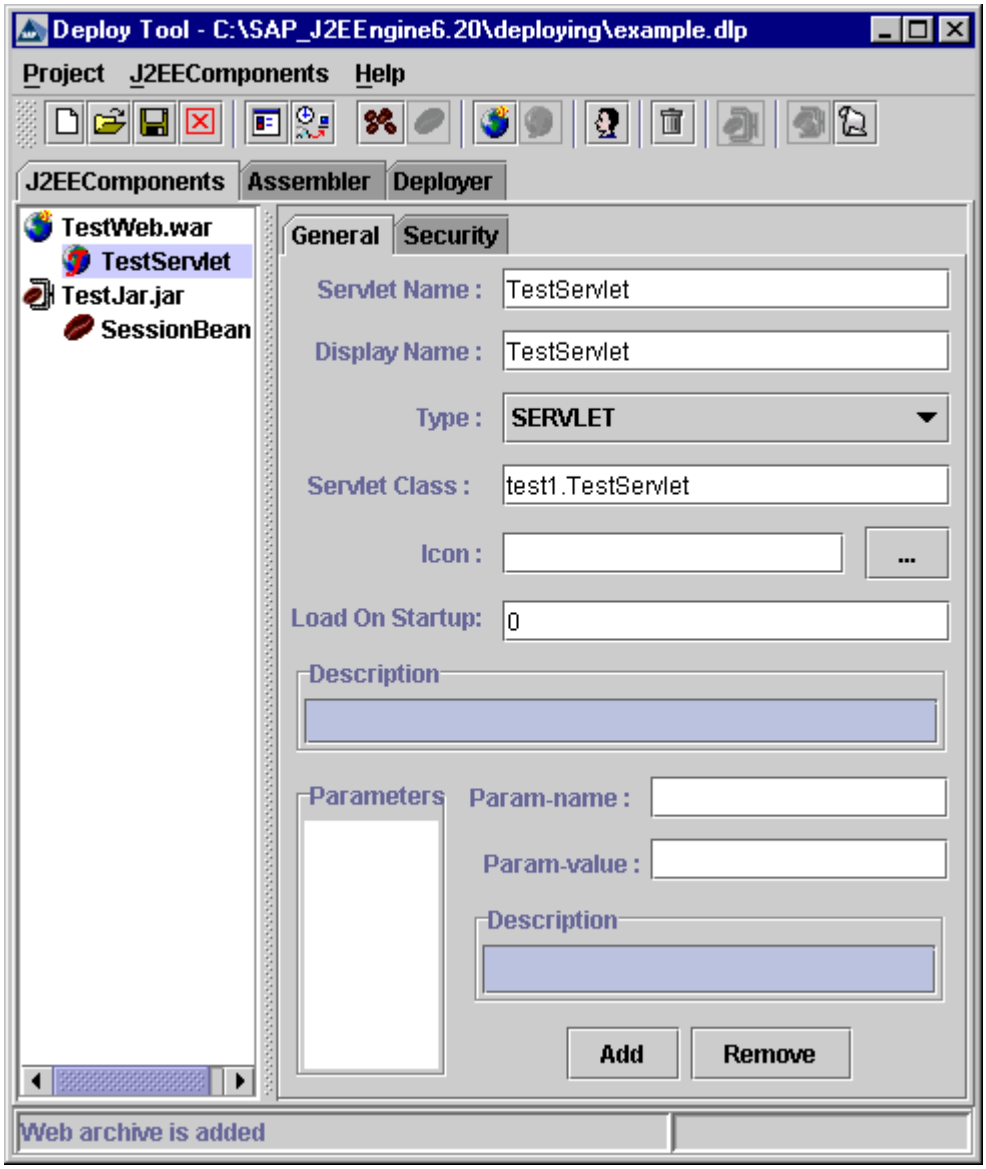
To add the Servlet or JSP from file, choose *J2EEComponents→Add Servlet/JSP→Add From Files. TestWeb.war* group Directories and Files nodes are scanned for Servlet classes and JSP files, and the names that best fit the Servlets or JSP are generated. You must select only the checkboxes that correspond to the files you want to add. Choose “Ok.”



Select the Servlet Class to Add

The Servlet appears as a subnode of the Web Archive in the tree structure of the “J2EEComponents” tab left-hand pane. Deploy Tool generates its properties automatically.

The example’s properties for *TestServlet* are shown below:

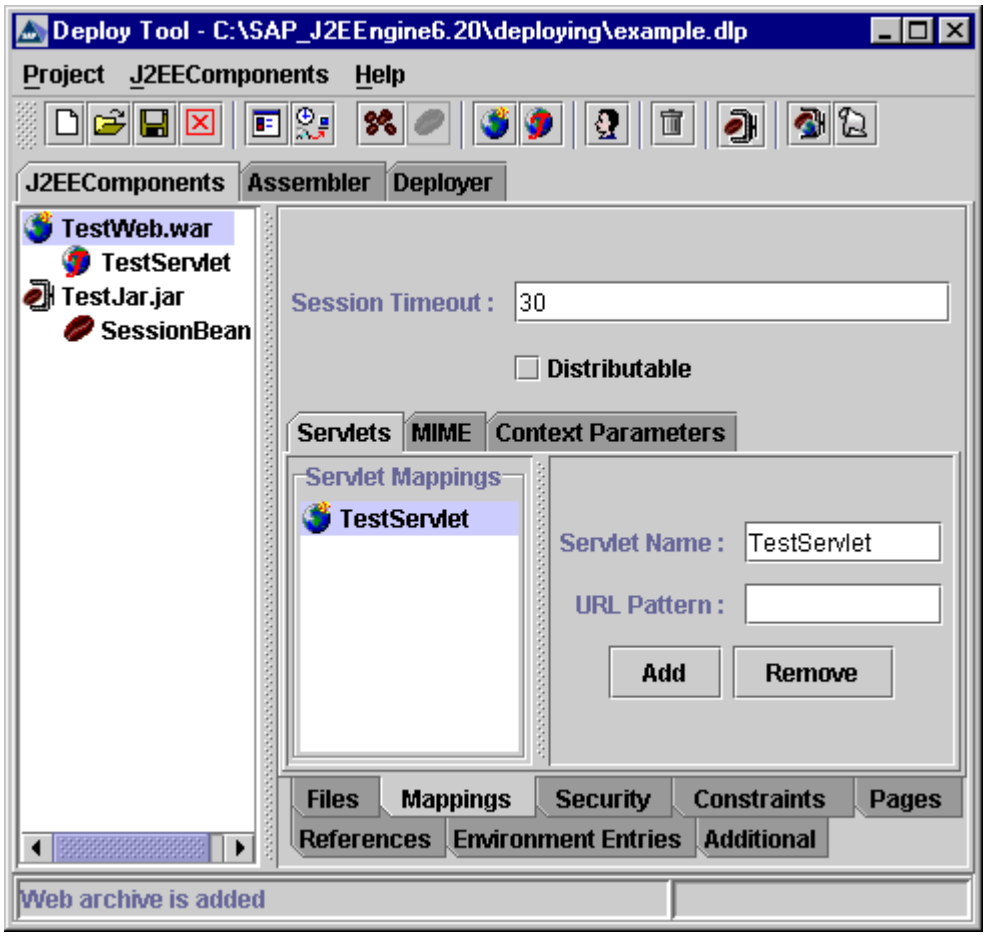


Servlet Properties

Step 10: Set Servlet Mappings

The Servlet container uses URL paths to map requests to Servlets. This is defined using the Servlet mapping elements, in which the “Servlet Name” and the “URL Pattern” must be specified.

Choose the “Mappings” tab from the “J2EEComponents” tab right-hand pane for a WAR. Choose the “Servlets” subtab and fill in the “Servlet Name.” Choose “Add.” The Servlet appears in the “Servlet Mappings” pane.



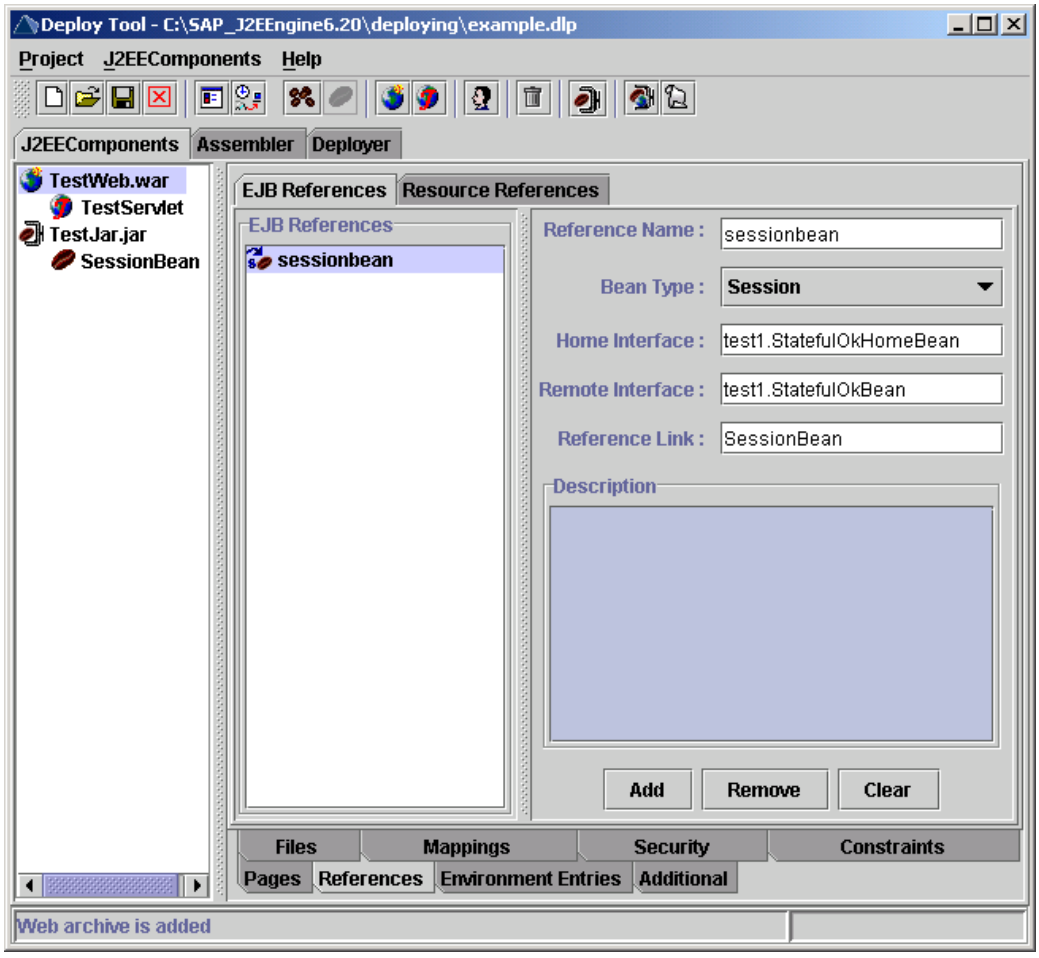
Servlet Mappings

This example does not require specific URL Pattern. Therefore, leave the “URL Pattern” field empty.

Step 11: Set EJB References

When a Web Archive uses Enterprise JavaBeans, you must set EJB References. In the current example, the *TestWeb.war* uses a Session EJB, named *SessionBean*.

To set the EJB references, choose the “References” tab on the right-hand pane of the “J2EEComponents” tab for a WAR. Select the “EJB References” subtab. Fill in the “Reference Name,” “Home Interface,” “Remote Interface,” and “Reference Link”, as shown below, and choose “Add.” “sessionbean” appears in the “EJB References” pane.



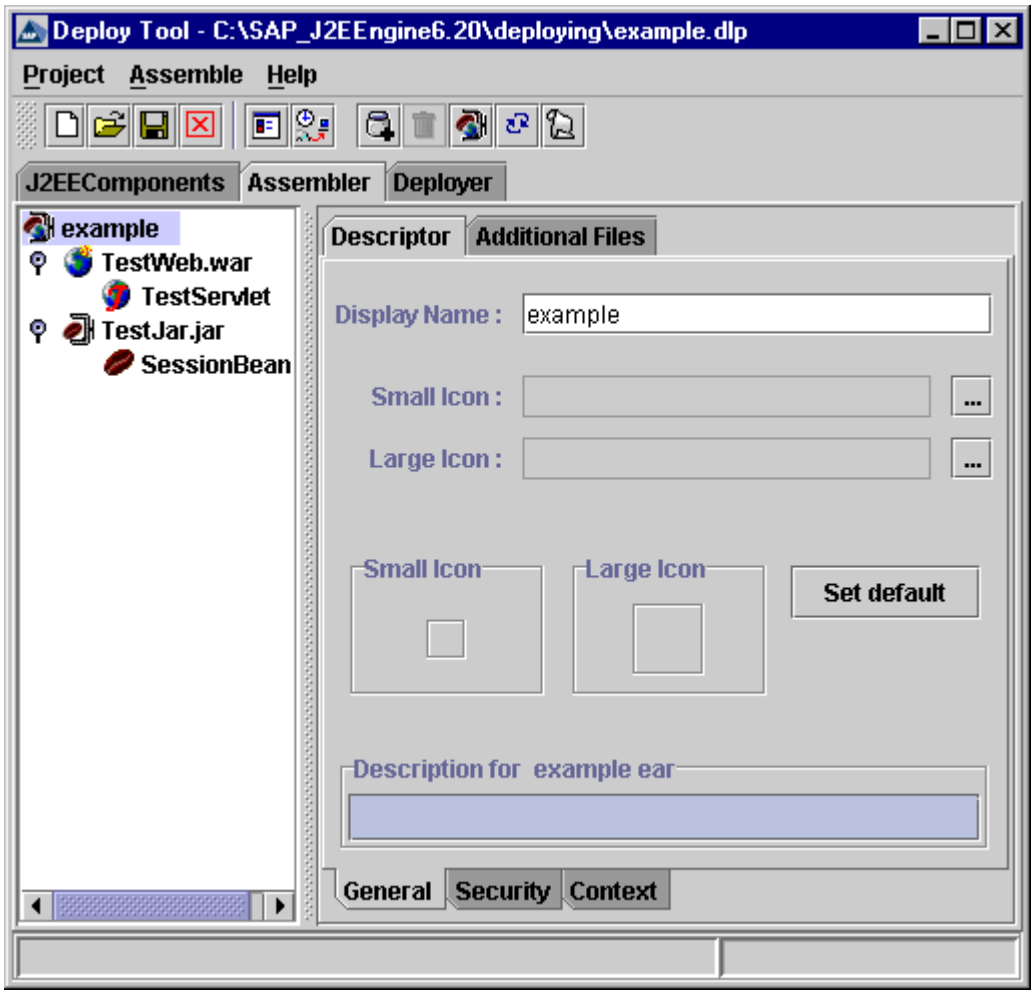
EJB References

The “Reference Name” must be the name the Servlet code uses to lookup the EJB. The “Reference Link” denotes that the reference is linked to an EJB in the J2EE application package. Its value must be the name of the EJB in the package. Choose “Add” to set this reference.

Step 12: Make the Archives

The J2EE components must be assembled in an application EAR. Choose the *J2EEComponents→Make All Archives,* or on the toolbar. The Deploy Tool status bar shows a message whether the archives are made successfully.

Choose the “Assembler” tab. If the archive was made successfully, the example tree structure appears in the left-hand pane of the “Assembler” tab.



The “Assembler” Tab

The “Display Name” field contains the name that is displayed by the graphical user interface tools. The value of the “Display Name” is set by default to be the name of the project file. This field must be given a unique name for the J2EE Engine, because the deployed application has this name in the J2EE Engine namespace.

Note: In the “Display Name”, do not use the following special symbols: slash (/), double backslash (\\), double quotation (“), colon (:), asterisk (*), question mark (?), backslash (\), greater than sign (>), less than sign (<) and vertical slash (|). They will be replaced with underscore (_). Do not end the name with period (.).

The “Assembler” tab may contain several archives (EJB JARs, Client JARs and WARs) that, according to the J2EE™ Specification, must be integrated in an application EAR.This EAR is used to deploy the application, so the next step is to make the EAR.

Step 13: Make the EAR

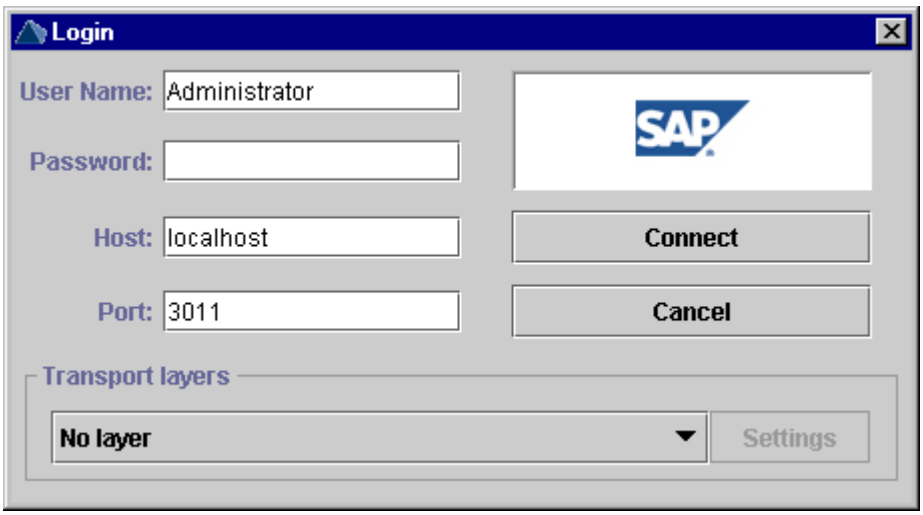
Select *Assemble→Make Ear,* or  on the toolbar in the “Assembler” tab. Enter EAR file name in the dialog box that appears, for example Test Example. Choose “Ok”

The message in the Deploy Tool status bar shows whether the EAR is made successfully.

Step 14: Connect to SAP J2EE Engine

To set further J2EE Engine-dependent properties for the application, a connection between the Deploy Tool and SAP J2EE Engine must be established. Therefore, the cluster properties must be specified correctly. Select the “Deployer” tab, choose *Deploy→Properties→LoginInfo* and set the user name, password, host and port. Before you start to deploy, you must have a SAP J2EE Engine (cluster or stand-alone version) already started.

To connect, select the “Deployer” tab; choose *Deploy→Connect*, or  on the toolbar. The “Login” dialog box appears. All property fields must be filled in correctly to establish a connection.



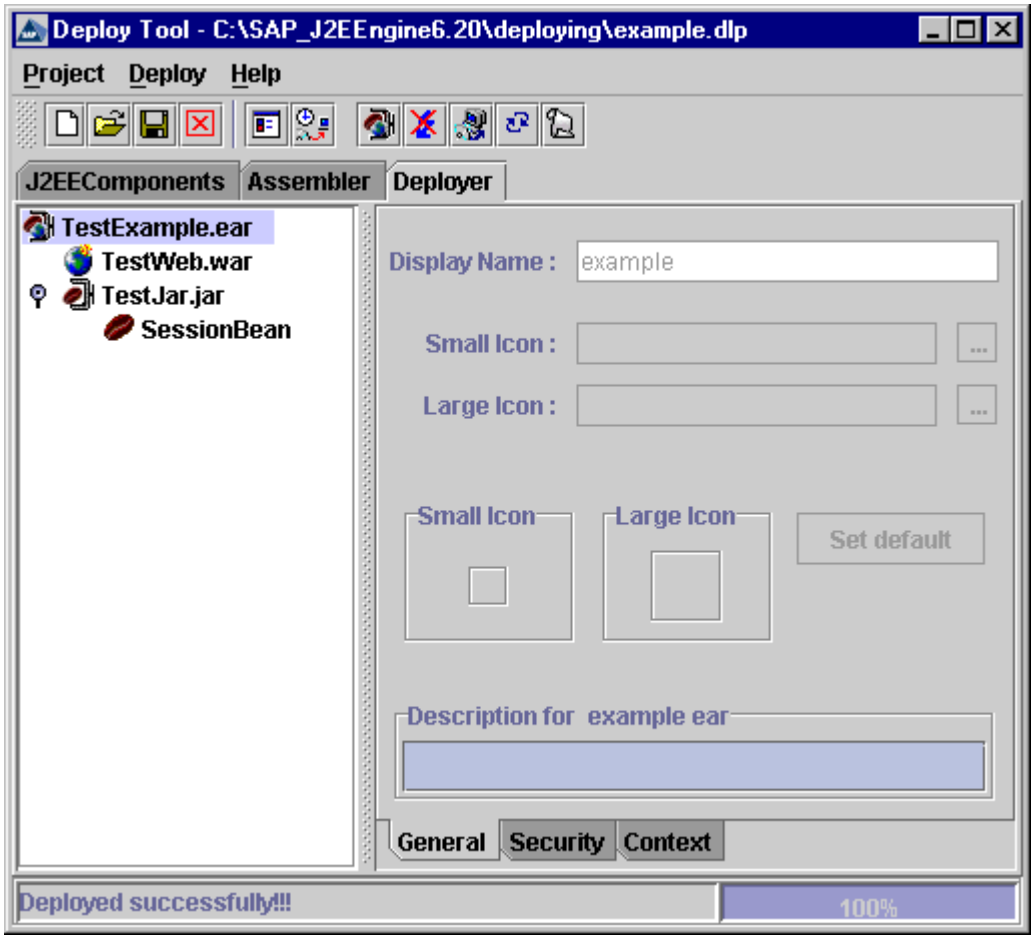
The “Login” Dialog Box

The “User Name,” “Password,” “Host,” “Port,” and “Transport Layers” fields are user dependent. Specify them according to your SAP J2EE Engine configuration. Then choose “Connect.” The Deploy Tool status bar shows whether the connection has been established. If an error occurs, a message dialog box appears.

The security properties can be set at this point. There are no specific security roles for the current project. Therefore, no Security Roles and User Mapping must be set.

Step 15: Deploy

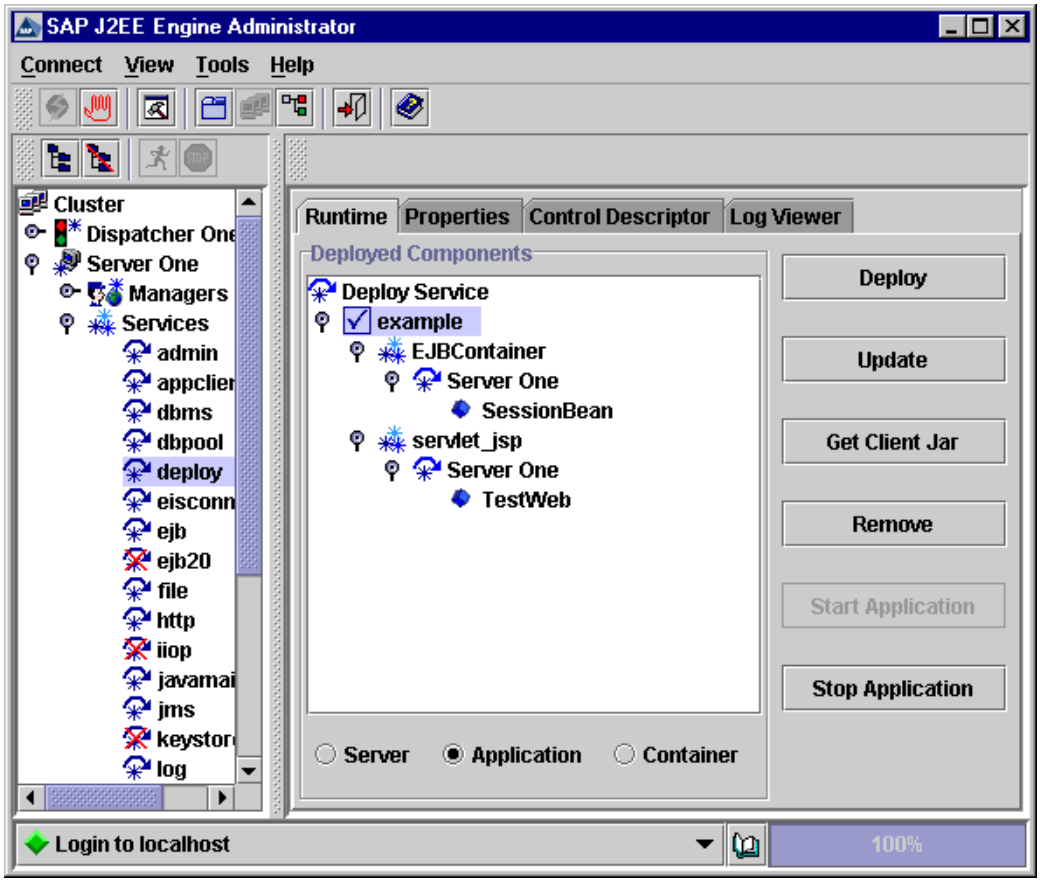
Choose *Deploy→Deployment→Deploy Ear*, or  on the toolbar. A “Start deploying ...” note appears in the left-hand part of the status bar. The progress bar shows the rate of the procedure execution. If deployment fails, an error message appears. The successfully completed deployment is shown below.



Deployed Successfully

Note: To deploy or re-deploy an application correctly on the J2EE Engine, complete the following additional requirement. The *work* subdirectories of all services (including the files within) must be closed and no other application can be permitted to read or write during the time of deployment. Otherwise, the deployment is performed, but a “Compiling Exception” appears when the application is started.

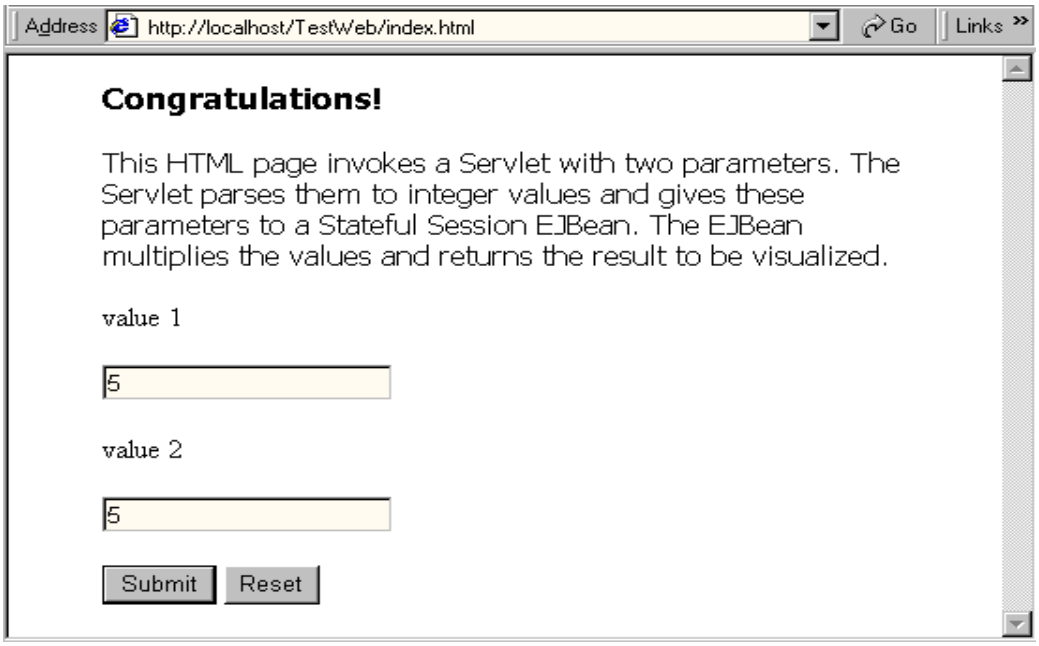
You can also check if the example is deployed correctly using the SAP J2EE Engine Visual Administrator. Connect the Visual Administrator to the J2EE Engine and check whether the application was deployed properly.



SAP J2EE Engine Deploy Service “Runtime” Tab

Run the Example

To see the result of the example application, use the following URL in a Web browser (Internet Explorer, Netscape, and so on.) – `http://localhost/TestWeb`, where "TestWeb" is the name of the WEB Archive set in *Step 6*. Enter integer values in the "value 1" and "value 2" fields, as shown below, and choose "Submit."



The Running Example

The Web browser displays the calculated result:

The Result

Chapter 4

Troubleshooting and Known Bugs

Troubleshooting and Known Bugs

Close All Work Sub-Directories and Services During Deploy Time

To correctly deploy or re-deploy an application, make sure you have closed all services and their *work* sub-directories on the server side, and no other application is admitted to read or write at the time of deployment. Otherwise, when starting the application, a “Compiling Exception” occurs.

This is a platform-dependant problem. When the services’ *work* sub-directories are opened in Windows Explorer, the Java programs cannot create or store files in them. No error messages are received or logged and the program assumes that the files are created, although they do not exist.

Do Not Restart the EJB or the Web Container During Deploy Time

When deploying, updating, undeploying, starting, or stopping an application, all containers must be running to perform the process requested. If the user restarts one of the containers (or both) during deployment, the process stops and the application is deleted. If one of the containers is stopped during the first stage of updating an application, all changes are removed and the initial status is recovered; during the last steps of updating – the whole application is removed.

Specifics Concerning the GUI Implementation

In Deploy Tool after modifying a property value of an existing entry in a table or a list, press “Set” or “Add” to confirm the changes prior to saving the project file. Otherwise, the changes will not be stored.

RMI/IIOP Support

If no support is set, the application will be deployed with P4. There is RMI/IIOP support for EJB application deployment but to use it, make sure the IIOP service is running both on dispatcher and server nodes.

SAP J2EE Engine Parser

All script files in the *<SAPj2eeEngine_install_dir>/deploying* directory use the SAP J2EE Engine parser for deployment. This is indicated by the `-Dserver.parser.inqmy=true` command in the script files.

The parser’s JAR file is *<SAPj2eeEngine_install_dir>/deploying/lib/inqmyxml.jar*. If you want to use a different parser, specify the classpath to the new parser’s JAR file and remove the `-Dserver.parser.inqmy=true` command from the starting script. The JAR must be created according to the JAXP specification.

“Already Started Operation with Application”

If you get this message, this means that one of the following actions over an application is performed in the cluster: deploy, update, stop, start, remove, or synchronize. During these operations no other changes are allowed over this application in the cluster. Wait until the action is over and try again.

Visual Administrator Deploy Service Runtime Tab Warning

If during a redeployment procedure you receive a warning message that your application is not found in the cluster, select another cluster node and after a few minutes try again to access the Deploy Service in the current node. This will refresh the information in the Visual Administrator.

J2EE Components Generator – Creating EJB JARs

Currently, for all classes and interfaces described in the EJB descriptors of a JAR, there is a simple dependency procedure that tries to find all additional files in the classpath that are not included in the JAR and includes them automatically. In some special cases this might cause some problems. Some files that are defined in a standalone library and deployed later as a library might also be included in the JAR. With the classloading system after 6.20 SP2, which first searches through the references and then in the application loader, there is a small chance to have a problem especially if the reference is defined before application deployment. In such a case you have to remove these additional classes manually from the archive. In the 6.30 release, the Deploy Tool will provide a way to check these dependencies and decide whether it is needed to include the files or not.