Chapter 14 of *WebSphere Application Server V7 Administration and Configuration Guide,* SG24-7615

WebSphere Application Server V7: Packaging Applications for Deployment

In this chapter we discuss packaging of Java™ Platform, Enterprise Edition 5 (JEE 5) applications using Enterprise JavaBeans™ 3.0 (EJB™ 3.0). We also provide a section on working with business-level applications, a new concept introduced in WebSphere Application Server v7.0.

We cover the packaging of the following JEE artifacts:

- Enterprise Archives (EAR)
- EJB 3.0 modules
- Web modules
- JPA persistence units
- Working with Enhanced EAR files

We also describe some of the IBM® enhancements WebSphere Application Server v7.0 supports in addition to the JEE specification. If you are working on a pre-JEE 5 application or are using EJB 2.1 or earlier modules, for details on the earlier versions, also refer to Chapter 13 in the Redbooks® publication, *WebSphere Application Server V6.1: System Management and Configuration*, SG24-7304, which is available at:

`http://www.redbooks.ibm.com/abstracts/sg247304.html?Open`

JEE 5 EAR files

WebSphere Application Server v7.0 supports the JEE 5 and the EJB 3.0 specifications, which make use of annotations. Because developers annotate the source code with information about how it should be deployed, much of the packaging and deployment tasks that were necessary in previous versions of WebSphere Application Server are no longer relevant. Annotations also reduce the number of classes and interfaces the developer needs to manage within the project. With the introduction of JEE 5 developing, packaging and deploying Java enterprise applications have been greatly simplified.

As with previous versions of the Java 2 Enterprise Edition (J2EE™) specification JEE 5 applications are packaged in Enterprise Archive (EAR) files. An EAR file can contain Web Archive (WAR) files, EJB modules (packaged as EJB JARs), Resource Adapter Archive (RAR) files, Java Utility Projects (packaged as JAR files), and Application Client modules. Figure 1 shows a schematic overview of a JEE EAR file.

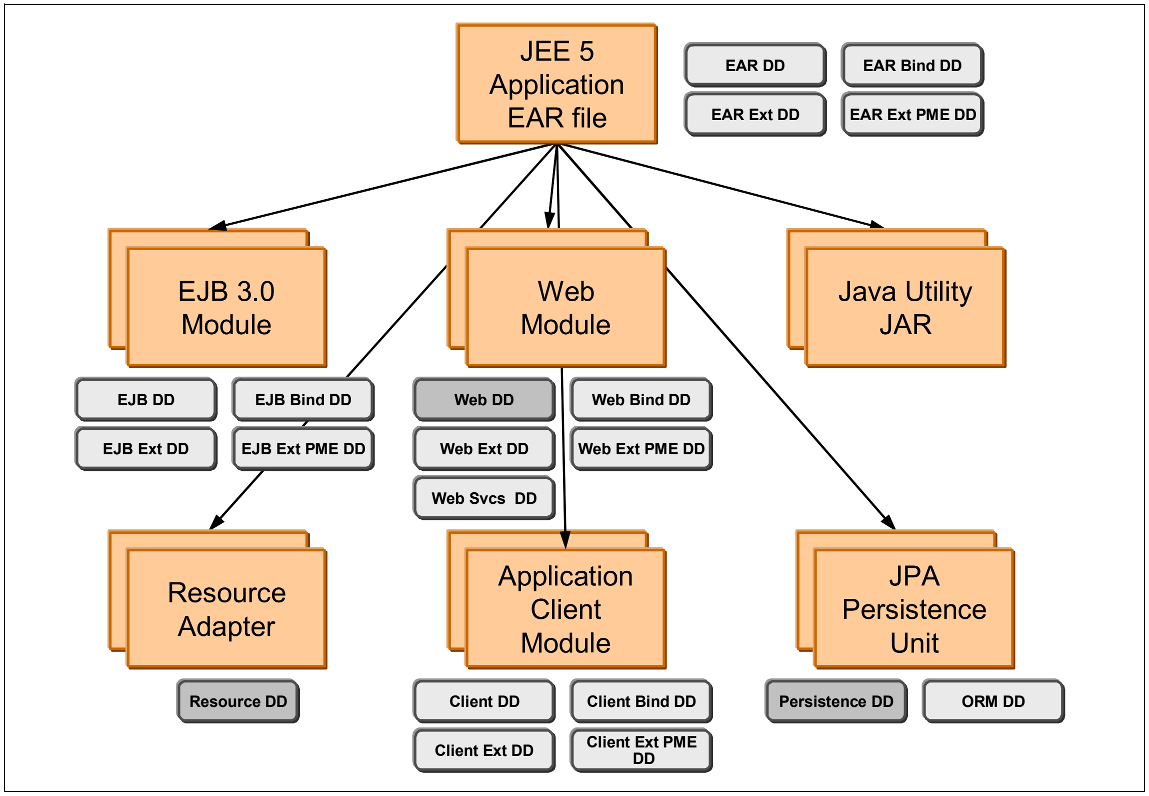


Figure 1 JEE 5 EAR file structure

One major difference between JEE 5 applications and previous J2EE applications is that, if the source code is properly annotated, in many cases the EAR file or modules do not need to contain any deployment descriptors. In previous J2EE versions, deployment descriptors were necessary to tell the application server how to deploy the modules and how clients should locate EJB interfaces, for example.

JEE 5, on the other hand, relies on default values, and as long as the default values are acceptable, you do not need to include a deployment descriptor. However, an optional deployment descriptor can be included and it will then override the defaults and the settings given by the annotations in the source code. This gives the deployer the flexibility to deploy the application as preferred for the target environment.

In Figure 1 on page 2 the required deployment descriptors are marked with light gray color and the optional ones with dark gray.

Table 1 lists the deployment descriptors valid for a WebSphere Application Server v7.0 EAR file.

Table 1 Enterprise Archive Deployment Descriptors

Filename	**Required**	**Content**
application.xml	No	Defines modules and security roles used by the enterprise application.
ibm-application-bnd.xml	No	Mappings for security roles.
ibm-application-ext.xml	No	WebSphere®-specific application extensions.
ibm-application-ext-pme.xml	No	Configuration for WebSphere's programming extensions to the JEE specification.

Development tools

The development tools Rational® Application Developer for WebSphere Software 7.5 (RAD) and Rational Application Developer Assembly and Deploy Features for WebSphere 7.0 (RAD-AD) provide editors for all the deployment descriptors.

RAD-AD comes with the WebSphere Application Server license and is a subset of the RAD product. It contains only the Eclipse plug-ins needed to develop, test, package, and deploy J2EE/JEE applications for WebSphere Application Server v7.0, but not the productivity enhancements features found in RAD. RAD-AD 7.0 also only supports testing on a WebSphere Application Server v7.0, while RAD 7.5 supports testing also on previous versions of WebSphere Application Server.

Working with deployment descriptors

To work with the deployment descriptors, start RAD or RAD-AD and create a new JEE application project, or import an existing. To create a new project select **File** → **New** → **Enterprise Application Project** and follow the wizards. To import an existing project select **File** → **Import...** and follow the wizards.

Then, in the Enterprise Explorer view, right-click the JEE project and select **Java EE** → **Generate Deployment Descriptor Stub**.

Expanding the JEE projects META-INF folder reveals the created `application.xml` deployment descriptor file. To edit it, either double-click the file or double-click the EAR file's deployment descriptor icon, as shown in Figure 2.

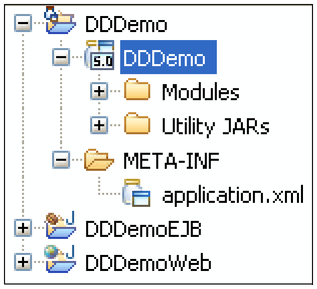
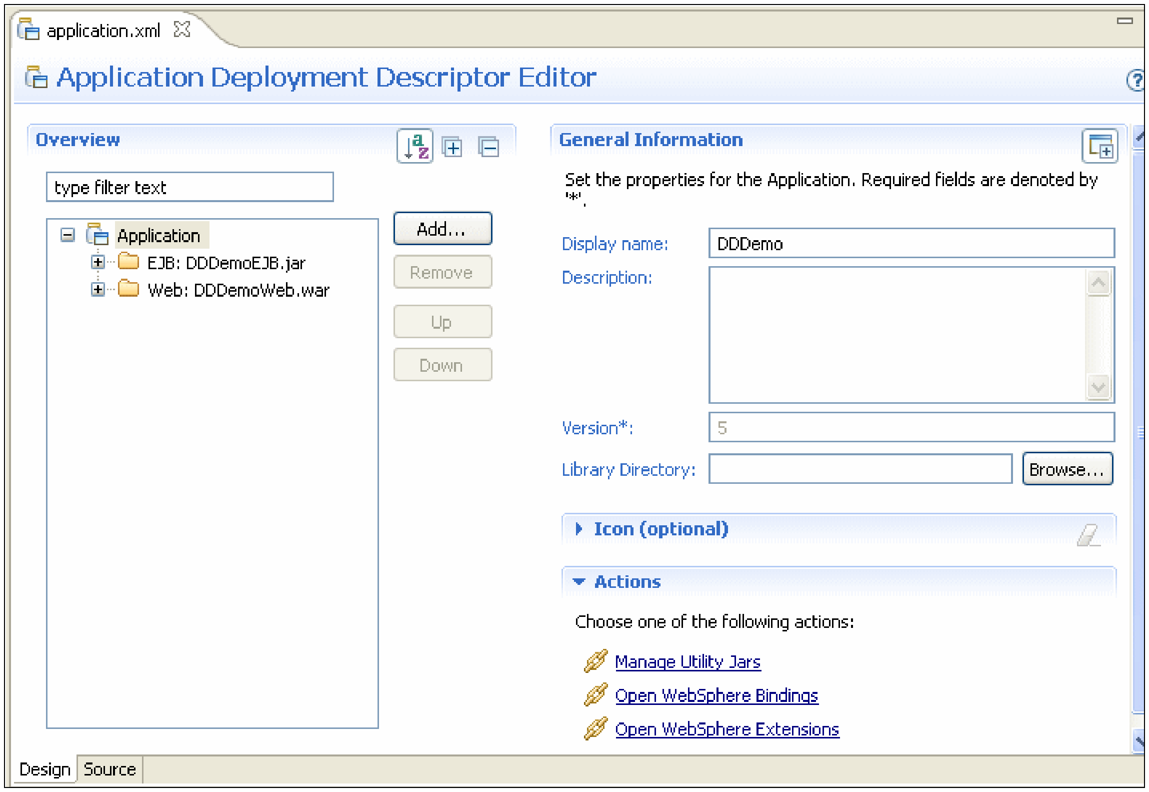


Figure 2 JEE EAR deployment descriptor icon in RAD-AD

On the right hand side of the deployment descriptor editor is a panel with fields for the information that can be entered into the deployment descriptor. See Figure 3. Fill out the fields and then press **Ctrl-S** to save the deployment descriptor.


application.xml

Application Deployment Descriptor Editor

Overview

type filter text

- Application
  - EJB: DDDemoEJB.jar
  - Web: DDDemoWeb.war

General Information

Set the properties for the Application. Required fields are denoted by "*".

Display name: DDDemo

Description:

Version*: 5

Library Directory:

Icon (optional)

Actions

Choose one of the following actions:

- Manage Utility Jars
- Open WebSphere Bindings
- Open WebSphere Extensions

Figure 3 JEE EAR deployment descriptor editor

To create any of the other optional JEE EAR-file level deployment descriptors, right-click the JEE EAR project. Select **Java EE** and then one of the remaining deployment descriptor options; **Bindings**, **Extensions**, or **Programming Model Extensions Deployment Descriptor**. Selecting any of these options creates the corresponding deployment descriptor file in the JEE EAR file's META-INF folder. The file can then be accessed either by double-clicking it in the META-INF folder, or by clicking the corresponding link under the Actions heading on the main JEE EAR deployment descriptor editor (Figure 3).

EJB 3.0 modules

EJB 3.0 modules are packaged similar to EJB 2.x modules. One major difference, however is that with EJB 3.0 there is no such thing as Container Manager Persistence (CMP) or Bean Managed Persistence (BMP) beans.

Instead, EJB 3.0 modules should use JPA for data persistence. EJB 2.x and earlier modules still support CMP and BMP beans as per the J2EE specifications and are still supported by WebSphere Application Server v7.0. Table 2 lists the deployment descriptors for an EJB 3.0 module.

Table 2 EJB 3.0 Deployment Descriptors

Filename	Required	Content
ejb-jar.xml	No	EJB and EJB method definitions, transaction attributes, resource references, and so on.
ibm-ejb-jar-bnd.xml	No	Explicit binding names for EJBs, and EJBs and resource references.
ibm-ejb-jar-ext.xml	No	Configuration of WebSphere extensions to the JEE EJB module specification.
ibm-web-ext-pme.xml	No	Configuration for WebSphere Programming Extensions to the JEE specification.

EJB interface bindings

WebSphere Application Server v7.0 binds EJB 3.0 interfaces and homes into two distinct JNDI namespaces, one JVM™-local and one global namespace. Local interfaces and homes are bound to the JVM-local namespace, and remote interfaces and homes are bound to the global namespace.

Unless overridden by explicitly assigned bindings, the interfaces are bound using default names generated automatically by the EJB container. Each default name has a short version and a long version. The short name consists of only the Java package name and class name of the interface. The long name prefixes the short name with a component ID, which is composed of the enterprise application name, the module name and the component name.

Consider an enterprise application called RAD75EJBWebEAR that has an EJB module called RAD75EJB.jar with the following bean and interfaces:

- A session bean with an implementation class called EJBBankBean
- A local interface called itso.bank.service.EJBBankService
- A remote interface called itso.bank.service.EJBBankRemote

The auto-generated short and long names for the bean's interfaces are:

- `ejblocal:itso.bank.service.EJBBankService`
- `ejblocal:RAD75EJBWebEAR/RAD75EJB.jar/EJBBankBean#itso.bank.service.EJBBankService`
- `itso.bank.service.EJBBankRemote`

- `ejb/RAD75EJBWebEAR/RAD75EJB.jar/EJBBankBean#itso.bank.service.EJBBankRemote`

The local names are bound into the JVM-local namespace called `ejblocal`. The remote names are bound to the global namespace, and to avoid cluttering the root of the namespace the long name is prefixed with `ejb/`.

The auto-generated default names can be overridden by placing a file named `ibm-ejb-jar-bnd.xml` in the EJB JAR module's META-INF directory with the preferred names. By overriding the default names you can define your own naming convention independently from how the beans are packaged into the application/module hierarchy.

EJB reference resolution using the AutoLink feature

When an EJB client (typically a servlet, or another EJB) wants to call an EJB, it first needs to locate the EJB home in the JNDI namespace. In EJB 2.1, and earlier, this had to be done with a few lines of code written explicitly by the EJB client developer. However, with the EJB 3.0 support and source code annotations WebSphere Application Server v7.0 uses a feature called AutoLink which automates this task in many cases, making the lookup code superfluous.

When the EJB container encounters an annotation for an EJB reference, it tries to automatically look up the referenced EJB. The AutoLink algorithm first looks to see if the EJB interface has been explicitly given a name in the module’s bindings file. If not found, AutoLink searches within the referring module for an EJB that implements the interface. If it does not find exactly one EJB that implements the interface within the same module, AutoLink expands the search scope and searches within other modules defined in the application. If it finds exactly one EJB that implements the interface, it uses that as the reference target.

The scope of AutoLink is limited to the enterprise application in which the EJB reference appears and within the application server on which the referring module is assigned. If the target EJB resides in an application other than the client’s, or it is deployed on an application server other than the client’s, then AutoLink does not work. In this case, target bindings must be explicitly defined in the client’s bindings file. For an EJB module, this is the `ibm-ejb-jar.bnd.xml` file, and for a Web module, it is the `ibm-web-bnd.xmi` file.

For more information and some examples of how to do this, refer to Chapter 9 in the Redbooks publication, *WebSphere Application Server Version 6.1 Feature Pack for EJB 3.0*, SG24-7611, which is available at:

http://www.redbooks.ibm.com/abstracts/sg247611.html?Open

The AutoLink feature only handles EJB references and is available for clients running in the EJB container, Web container, or application client container.

Note: If you installed the Feature Pack for EJB 3.0 WebSphere Application Server v6.1, the default was to scan annotations during the installation of an EJB 3.0 module. For WebSphere Application Server v7.0, the default is not to scan pre-Java EE 5 modules during the application install or at server startup.

To preserve backward compatibility with both the Feature Pack for EJB 3.0 and the Feature Pack for Web Services, you have a choice whether or not to scan legacy Web modules for additional metadata. A server level switch is defined for each feature pack scan behavior. If the default is not appropriate, the switch must be set on each server and administrative server that requires a change in the default. The switches are the server custom properties com.ibm.websphere.webservices.UseWSFEP61ScanPolicy={true|false} and com.ibm.websphere.ejb.UseEJB61FEPScanPolicy={true|false}.

Go to **Servers** → **Server Types** → **WebSphere application servers** → ***server_name*** → **Java and Process Management** → **Process Definition** → **Java Virtual Machine** → **Custom Properties**, and add a new property.

Just-In-Time generation of EJB deployed code

EJB modules must contain EJB deployed code in order for an application server to be able to run the EJBs. EJB deployed code contains application server specific code that bridges the EJB interface and implementation code to the application server's EJB implementation.

In previous versions of WebSphere Application Server and with previous versions of J2EE, the EJB deployed code could be generated using three methods:

- During development, using the Prepare for Deploy action in Rational Application Developer or the Application Server Toolkit
- Before installing an EAR file to WebSphere Application Server, using the EJBDeploy tool from command line
- During installation of an EAR file to WebSphere Application Server, using the install panels in the administrative console

WebSphere Application Server v7.0 and the EJB 3.0 support introduces a new feature called Just-In-Time deployment. This feature removes the need to process the EJB modules to generate the deployed code. Instead, the EJB container dynamically generates the necessary code in-memory as needed when the application is running. This feature simplifies and speeds up the development, packaging, and deployment of EJBs.

For EJB 3.0 clients that are not running inside a Web container, EJB container, or client container that has been upgraded to the EJB 3.0 level, the Just-In-Time development does not generate the necessary classes. In this case, the `createEJBStubs` tool should be used and the generated classes would be made available on the client's classpath.

An example scenario where this is applicable is a servlet running in WebSphere Application Server v6.1 calling an EJB 3.0 bean running in WebSphere Application Server v7.0. In this case, the EJB stubs should be created manually and the generated classes added to the servlet's Web module.

For details and syntax on the createEJBStubs tool, refer to the WebSphere Application Server v7.0 Information Center at:

`http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/topic/com.ibm.websphere.nd.multiplatform.doc/info/ae/ae/rejb_3stubscmd.html`

Mixing different EJB versions in an EAR file

WebSphere Application Server v7.0 allows you to mix EJB 1.1, 2.x and 3.0 beans in the same JEE 5 enterprise application. EJB 1.1 and 2.x beans can be directly carried forward in EJB 1.1 and EJB 2.x modules respectively. WebSphere can also handle EJB 1.1, 2.x and 3.0 session beans, and EJB 2.x message driven beans within the same EJB 3.0 module. CMP and BMP entity beans are not supported in EJB 3.0 modules and must remain in EJB 1.1 or EJB 2.x modules.

JPA persistence units

Persistence units using Java Persistence API (JPA) can be packaged either into the module that uses the persistence unit or in a separate utility JAR file (packaged as a standard .jar file). If packaged as a separate utility JAR file, it must be referenced from the module that uses the persistence unit using the module's META-INF.MF Class-path directive.

Persistence units require a `persistence.xml` file, which defines a JPA entity manager's configuration. Among other information, the persistence.xml file lists the entity classes and the data source to use.

A persistence unit can also include an optional `orm.xml` file which specifies the object-relational mapping configuration. The orm.xml file is an alternative to using annotations and can be used to override annotations in the source code to specify how the objects should be persisted to the database.

Table 3 lists the deployment descriptors valid for a JPA persistence unit.

Table 3 JPA Persistence Units Deployment Descriptors

Filename	Required	Content
persistence.xml	Yes	Entity manager’s configuration, entity classes, data sources, and so on.
orm.xml	No	Object-to-relational mapping annotation overrides.

JPA access intents

WebSphere Application Server provides an optimization enhancement for EJB 2.x entity beans called access intents. However, because the EJB 3.0 specification does not support entity beans, the access intents support is not available for EJB 3.0 beans. Instead, WebSphere Application Server v7.0 provides JPA access intents which can be used to improve performance and scalability for JPA applications.

JPA access intents specify the isolation level and lock level used when reading data from a data source. JPA access intents can be used, providing that the following restrictions are honored:

- Access intent is available for the application in the Java EE server environment
- Access intent is applicable to non-query entity manager interface methods. Query should use query hint interface to set its isolation and read lock values.
- Access intent is only available for DB2® databases.
- Access intent is in effect only when pessimistic lock manager is used. To specify a pessimistic lock manager add the following statement to the persistence unit's property list:

  ```
  <property name="openjpa.LockManager" value="pessimistic"/>.
  ```

Table 4 compares EJB 2.x access intents to JPA access intents.

Table 4 JPA Access Intents properties

EJB 2.x entity bean access intent	JPA access intent	Description
optimistic	isolation: Read Committed	Data is read but no lock is held. Version id is used on update to insure data integrity. Other transactions can read and update data.
	lockManager: Optimistic	
	query Hint: ReadLockMode: READ	

EJB 2.x entity bean access intent	JPA access intent	Description
pessimistic read	isolation: Repeatable Read lockManager: Optimistic query Hint: ReadLockMode: READ	Data is read with shared locks. Other transactions attempting to update data are blocked.
pessimistic update	isolation: Repeatable Read lockManager: Pessimistic query Hint: ReadLockMode: WRITE	Data is retrieved with update or exclusive lock. Other writes are blocked until commit. This access intent can be used to serialize update access to data when there are multiple writers.
pessimistic exclusive	isolation: Serializable lockManager: Pessimistic query Hint: ReadLockMode:WRITE	Data is retrieved with update or exclusive lock. Other writes are blocked until commit. This access intent can be used to serialize update access to data when there are multiple writers.

JPA access intents are specified in the `persistence.xml` deployment descriptor.

For more information about access intents, search the Information Center for JPA Access Intent.

For information about EJB 2.x access intents, see Section 13.5 in the Redbooks publication, *WebSphere Application Server V6.1: System Management and Configuration*, SG24-7304, available at:

http://www.redbooks.ibm.com/abstracts/sg247304.html?Open

Resource adapters

A resource adapter archive (RAR) module, also called a connector module, contains code that implements a library for connecting with a back-end Enterprise Information System (EIS), such as CICS®, SAP®, and PeopleSoft®. RAR files (called connectors) are packaged as a standard Java Archive with a .rar file extension. A resource adapter can be installed as a stand alone adapter or as part of an enterprise application, in which case the resource adapter is referred to as an embedded adapter.

A connector module contains a mandatory deployment descriptor file named `ra.xml`, residing in the module's META-INF directory.

Web modules

JEE 5 Web modules are packaged just like Web modules in earlier J2EE versions. A Web module can contain servlet code, JSPs, static HTML pages, images, JavaScript™, stylesheets, and so on.

A common challenge when working with Web modules is to make sure the right version of a required Java library is loaded. Often Web application developers need to include specific third party libraries such as log4j, or Xalan/Xerces, and must make sure that the correct version of a library is loaded for an application. This requires knowledge on how the EAR and Web module's class loaders work. Refer to Chapter 13, *Understanding class loaders* for detailed information about this topic.

A Web module supports several deployment descriptors, as shown in Table 5.

Table 5 Web module deployment descriptors

Filename	Required	Purpose
web.xml	Yes	Servlet definitions, URL mappings, and init parameters, servlet listeners, and so on.
ibm-web-bnd.xml	No	Mapping of logical resources used by the Web module to their runtime managed resources.
ibm-web-ext.xml	No	Configuration of WebSphere extensions to the JEE Web module specification.
ibm-web-ext-pme.xml	No	Configuration for WebSphere Programming Extensions to the JEE specification.
webservices.xml	No	Configuration of Web services, and implementation code.

Note: If an `application.xml` deployment descriptor is not included in the EAR file, the context root for a Web module defaults to the Web module's name without the .war extension.

WebSphere extensions to Web modules

WebSphere Application Server provides multiple extensions for Web modules. These are configured in the `ibm-web-ext.xml` deployment descriptor in the Web module. To create this file in RAD or RAD-AD, right-click the Web module in the Enterprise Explorer view and select **Java EE** → **Generate WebSphere Extensions Deployment Descriptor**. To edit the file, either expand the Web module's WebContent/WEB-INF folder and double-click the **ibm-web-ext.xml** file, or click the **Open WebSphere Extensions** link on the Web module's deployment descriptor editor for the web.xml file, as shown in Figure 4.

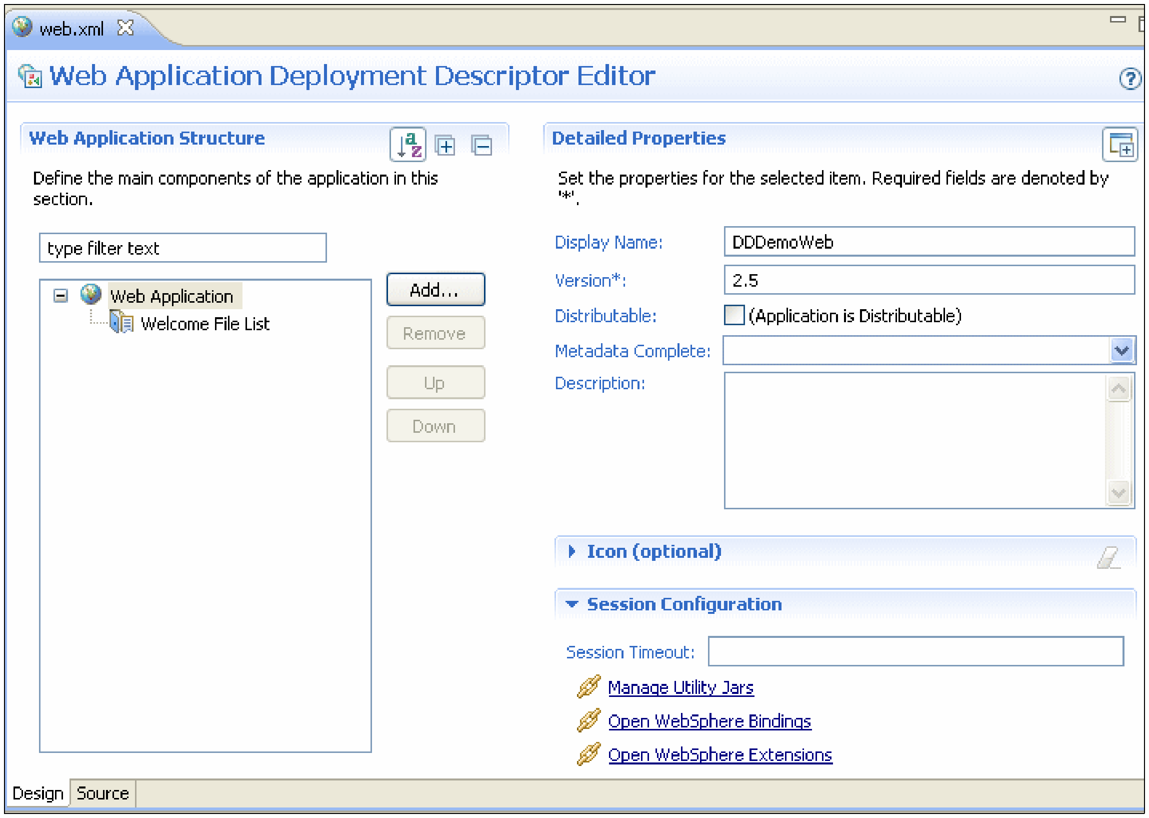
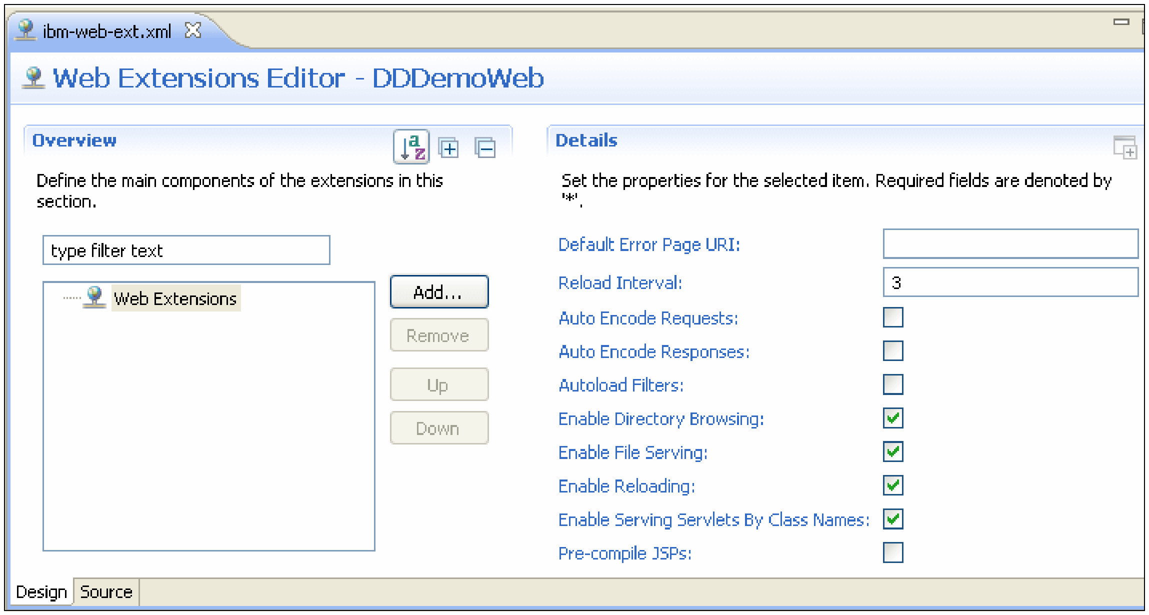


Figure 4   Web module's deployment descriptor editor

The Web module extensions editor is shown in Figure 5.


ibm-web-ext.xml
Web Extensions Editor - DDDemoWeb
Overview
Define the main components of the extensions in this section.
type filter text
Web Extensions
Add...
Remove
Up
Down
Details
Set the properties for the selected item. Required fields are denoted by "*".
Design
Source

Default Error Page URI:	
Reload Interval:	3
Auto Encode Requests:	☐
Auto Encode Responses:	☐
Autoload Filters:	☐
Enable Directory Browsing:	✓
Enable File Serving:	✓
Enable Reloading:	✓
Enable Serving Servlets By Class Names:	✓
Pre-compile JSPs:	☐

Figure 5 Editing WebSphere Web module extensions

Note: Previous versions of WebSphere Application Server provided a mechanism to share HTTP sessions across Web modules. Because this mechanism is not compliant with the servlet API specification, it had to be explicitly enabled for applications that required it. In WebSphere Application Server v7.0, this mechanism has been deprecated. If you have used this feature, you should redesign your application so that sessions are scoped at the Web module instead. If data really must be shared across Web module boundaries, use the IBM-enhanced session object, IBMApplicationSession, instead. You can search the Information Center for IBMApplicationSession for information about this.

File serving

When dealing with static content (HTML pages, images, style sheets, and so on), you can choose to have these resources served by WebSphere, or have them served by the HTTP server itself.

If you want WebSphere to serve the static content of your application, you must enable file serving. Enabling this activates a servlet which serves up any resource file packaged in the WAR file. The File serving enabled attribute is set to true by default. By changing it to false, the Web server plug-in will not send requests for static content to WebSphere, but leave it up to the HTTP server to serve them.

If you want the Web server to serve static content, you can experience better performance than using WebSphere in this instance, because the Web server is serving the content directly. Moreover, a Web server has more customization options than the file servlet can offer.

However, using the WebSphere file serving servlet has the advantage of keeping the static content organized in a single, deployable unit with the rest of the application. Additionally, this allows you to protect the static pages using WebSphere security.

To enable this option, check the **Enable File Serving** box.

Web application auto reload

If you check the **Enable Reloading** option, the class path of the Web application is monitored and all components, JAR or class files, are reloaded whenever a component update is detected. The Web module’s class loader is shut down and restarted. The **Reload Interval** is the interval between reloads of the Web application. It is set in seconds.

The auto reload feature plays a critical role in hot deployment and dynamic reload of your application.

Important: You must set the Enable Reloading enabled option to true for JSP™ files to be reloaded when they are changed on the file system. Reloading a JSP does not trigger the reload of the Web module, because separate class loaders are used for servlets and JSP.

This option is enabled by default, with the reload interval set to three (3) seconds. This means that classloader checks the classes on the classpath for updates every 3 seconds, and if any changes are found those classes are reloaded. But unless changes are detected nothing should happen to the classloader or the classes loaded. In production mode, you might consider making the reload interval much higher.

Serve servlets by class name

The invoker servlet can be used to invoke servlets by class name. Note that there is a potential security risk with leaving this option set in production. It should be seen as more of a development-time feature, for quickly testing your servlets.

A better alternative than this option is to define servlet mappings in the Web deployment descriptor for the servlets that should be available.

The invoker servlet is configured by the Enable Serving Servlets By Class Names option.

Default error page

This page will be invoked to handle errors if no error page has been defined, or if none of the defined error pages matches the current error.

Directory browsing

This Boolean defines whether it is possible to browse the directory if no default page has been found.

This option should be turned off for improved security.

Pre-compile JSPs

When a JSP is hit for the first time, it is automatically compiled into a servlet and then executed. To avoid this performance penalty the first time a JSP is hit, WebSphere allows JSPs to be pre-compiled during application installation instead of at first invocation. Selecting this option will cause the installation of the application to WebSphere to take longer, but the JSPs will be served faster on the first hit.

Automatic HTTP request and response encoding

The Web container no longer automatically sets request and response encodings and response content types. The programmer is expected to set these values using the methods available in the Servlet 2.4, and later, API. If you want the application server to attempt to set these values automatically, check the **Auto Encode Requests** option in order to have the request encoding value set. Similarly, you can check the **Auto Encode Responses** option in order to have the response encoding and content type set.

The default value of the autoRequestEncoding and autoResponseEncoding extensions is false, which means that both the request and response character encoding is set to the Servlet 2.4 specification default of ISO-8859-1. Different character encodings are possible if the client defines character encoding in the request header, or if the code uses the setCharacterEncoding(String encoding) method.

If the autoRequestEncoding value is set to true, and the client did not specify character encoding in the request header, and the code does not include the setCharacterEncoding(String encoding) method, the Web container tries to determine the correct character encoding for the request parameters and data.

The Web container performs each step in the following list until a match is found:

1. Looks at the character set (charset) in the Content-Type header.
2. Attempts to map the server's locale to a character set using defined properties.
3. Attempts to use the DEFAULT_CLIENT_ENCODING system property, if one is set.
4. Uses the ISO-8859-1 character encoding as the default.

If you set the autoResponseEncoding value to true and the following conditions are true:

- The client did not specify character encoding in the request header.
- The code does not include the setCharacterEncoding(String encoding) method.

Then the Web container performs the following actions:

- It attempts to determine the response content type and character encoding from information in the request header.
- It uses the ISO-8859-1 character encoding as the default.

Example: Packaging an application

As an example of how to package a JEE 5 application using EJB 3.0 beans, we use the same ITSOBank application developed by the team who wrote the Redbooks publication, *Rational Application Developer V7.5 Programming Guide*, SG24-7672. To download the sample application, go to:

`http://www.redbooks.ibm.com/redpieces/abstracts/sg247672.html?Open`

From there, click the **Additional Material** link. Then download the file `7672codesolution.zip` and unpack it to a directory on your computer. Unpacking the ZIP file creates a number of directories. The directory that we are interested in is the `ejb` directory. This directory contains two ZIP files with RAD project interchange files. We will now import both files into RAD-AD:

1. Start RAD-AD.
2. To import the code select **File** → **Import...** Expand the **Other** section and select **Project Interchange™**. Click **Next**.
3. Click the **Browse** button next to the From zip file field and browse to the `ejb` directory where you unzipped the sample code. Select the **RAD75EJB.zip** file and click **Open**.
4. Click the **Select All** button to select all projects in the file.
5. Then click **Finish**.
6. Repeat the process for the **RAD75EJBWeb.zip** project.

When the project files have been imported, the Enterprise Explorer view in RAD should look like Figure 6.

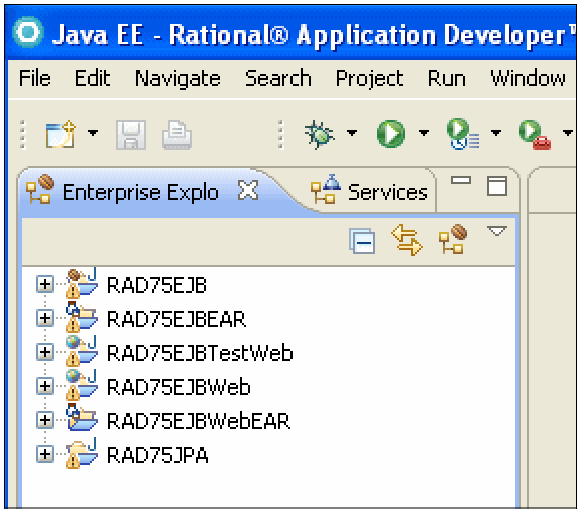


Figure 6   ITSO Bank application imported into RAD-AD

The Problems view shows 15 Warnings, but none of them are critical.

The workspace now has two Enterprise Application (EAR) projects, called RAD75EJBEAR and RAD75EJBWebEAR. RAD75EJBEAR contains some EJBs and a simple servlet for testing (in the RAD75EJBTestWeb project). A more sophisticated Web application is available in the RAD75EJBWeb project, and this is the one we will use in our example.

The RAD75EJBWeb project uses the EJBs in the RAD75EJB project, which in turn relies on the Persistence Unit in the RAD75JPA project.

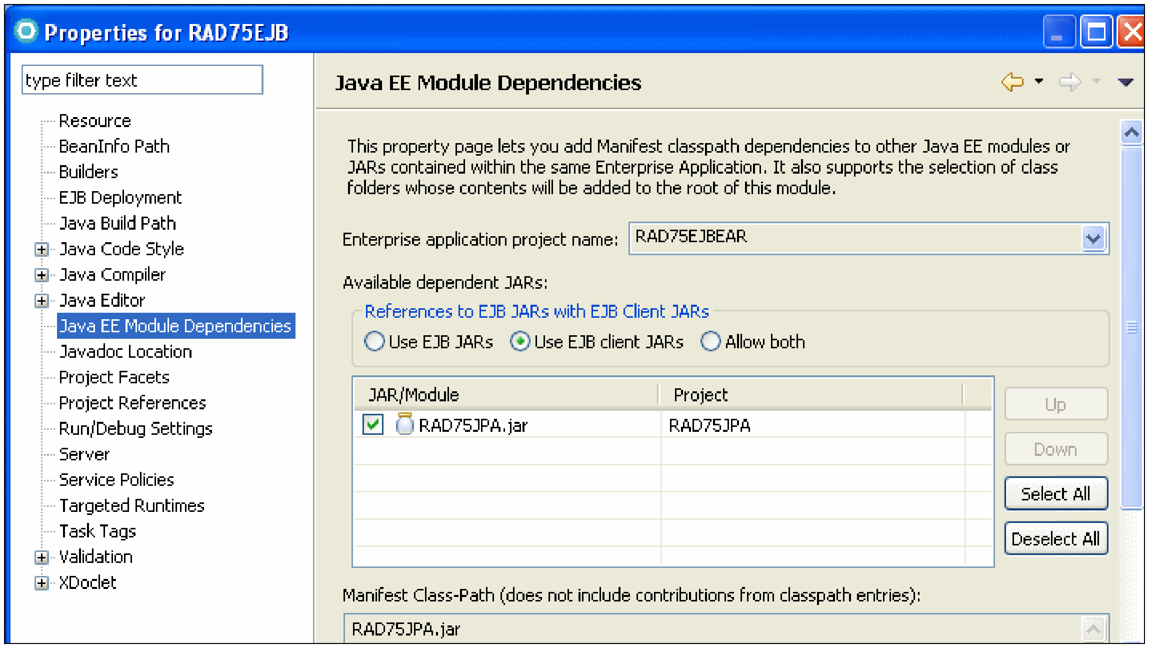
We will customize the RAD75EJBWeb project a little and export it as an EAR file The EAR file will be deployed in Chapter 15: *Deploying applications*.

None of the actions that we do here are actually necessary for getting the ITSO Bank application to work, because the development team has done everything necessary in the Redbooks publication, *Rational Application Developer V7.5 Programming Guide*, SG24-7672. However, to show you some common packaging tasks and the functionality of the RAD-AD development tool, we do some customizations to the application.

The first thing we will is to remove the unnecessary deployment descriptors, which were included in the application by the development team.

- Expand the **`RAD75EJB`** project and expand **`ejbModule`**. Then expand the **`META-INF`** folder and delete the **`ejb-jar.xml`** file.
- Expand the **`RAD75EJBWebEAR`** project and expand its **`META-INF`** folder. Delete the **`application.xml`** file.
- Expand the **`RAD75JPA`** project and expand **`src`**. Then expand the **`META-INF`** folder and delete the **`orm.xml`** file.

The RAD75EJB project depends on the Persistence Unit defined in the RAD75JPA project. To verify that this dependency is correctly set up, right-click the **RAD75EJB** project and select **Properties**. Then click **Java EE Module Dependencies** in the left pane. The right pane shows that the RAD75JPA.jar project is selected, which means that the EJB project can access the classes in the JPA project. See Figure 7.



Java EE Module Dependencies

This property page lets you add Manifest classpath dependencies to other Java EE modules or JARs contained within the same Enterprise Application. It also supports the selection of class folders whose contents will be added to the root of this module.

Enterprise application project name: RAD75EJBEAR

Available dependent JARs:

References to EJB JARs with EJB Client JARs

Use EJB JARs [✓ Use EJB client JARs] Allow both

JAR/Module	Project
✓ RAD75JPA.jar	RAD75JPA

Manifest Class-Path (does not include contributions from classpath entries):

RAD75JPA.jar

Figure 7 Java EE Module Dependencies

Configuring Web module extensions

For our application we customize the WebSphere Web module extensions:

1. Expand the **RAD75EJBWeb** project and double-click the **RAD75EJBWeb** heading (Figure 8) to open the Web module deployment descriptor.

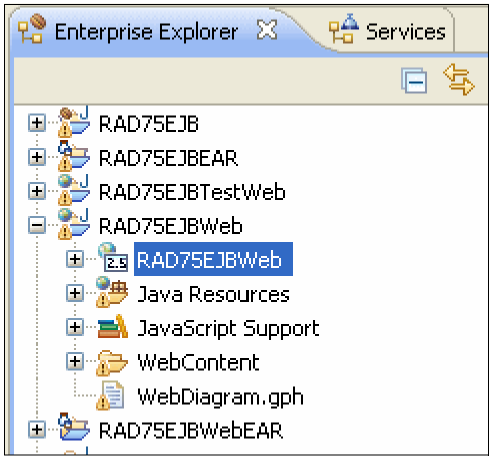
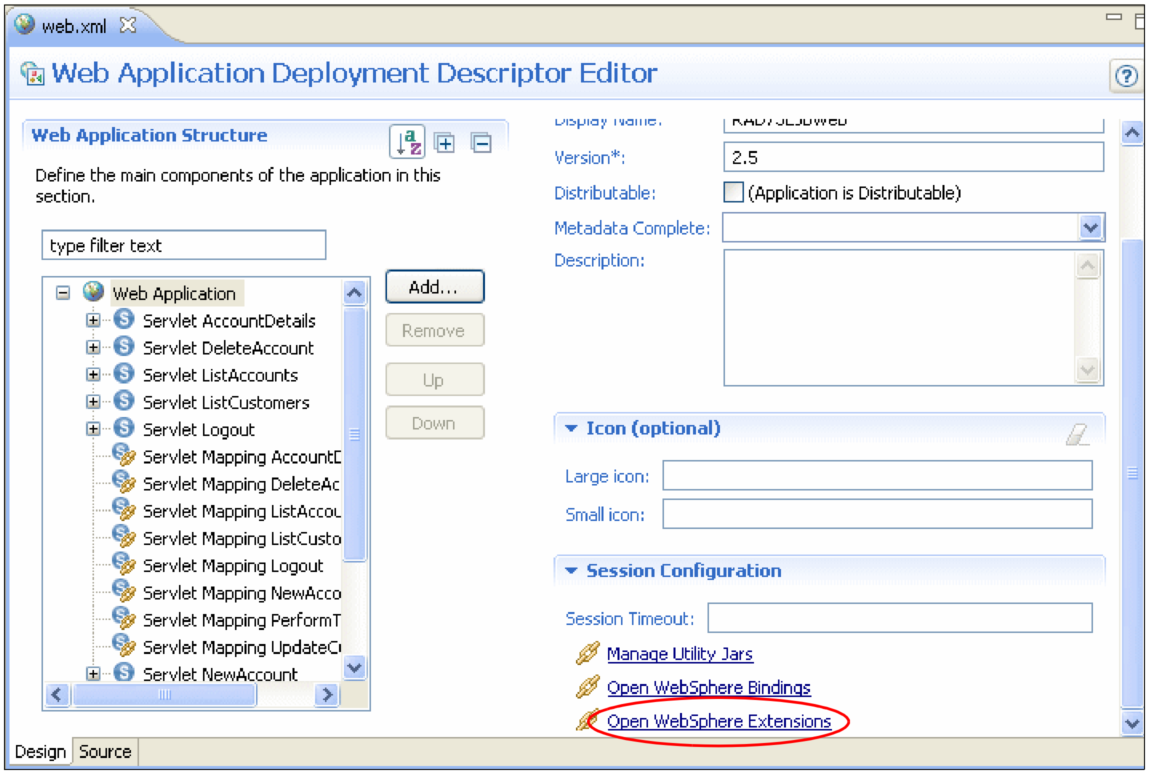


Figure 8   Opening the Web module deployment descriptor

2. When the panel opens, click the **Open WebSphere Extensions** link in the bottom right corner as shown in Figure 9 to open the extensions editor.


web.xml

Web Application Deployment Descriptor Editor

Web Application Structure

Define the main components of the application in this section.

type filter text

Web Application
Servlet AccountDetails
Servlet DeleteAccount
Servlet ListAccounts
Servlet ListCustomers
Servlet Logout
Servlet Mapping AccountD
Servlet Mapping DeleteAc
Servlet Mapping ListAccou
Servlet Mapping ListCusto
Servlet Mapping Logout
Servlet Mapping NewAcco
Servlet Mapping PerformT
Servlet Mapping UpdateC
Servlet NewAccount

Add...
Remove
Up
Down

Version*: 2.5
Distributable: (Application is Distributable)
Metadata Complete:
Description:

Icon (optional)
Large icon:
Small icon:

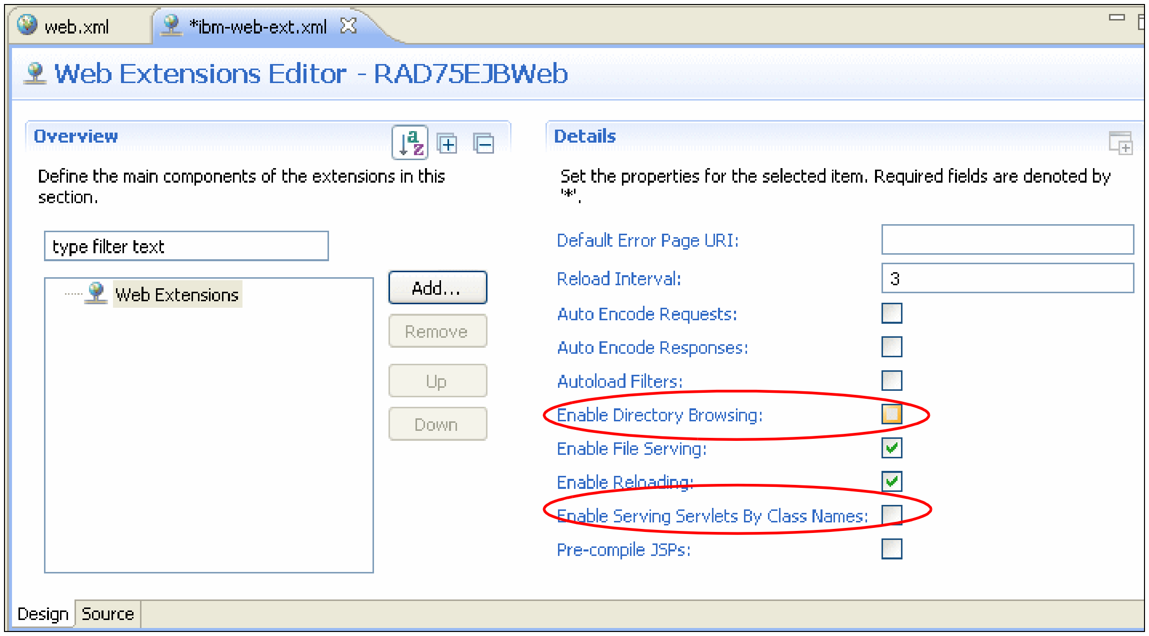
Session Configuration
Session Timeout:

Manage Utility Jars
Open WebSphere Bindings
Open WebSphere Extensions

Design
Source

Figure 9   Web module deployment descriptor editor

3. The Web module extensions editor contains options to configure the optional WebSphere extensions to Web modules. In our application, we do not want to serve servlets by classname and we also want to prevent directory browsing.
4. Therefore, we uncheck the corresponding check boxes as shown in Figure 10. It is a best practice to disable these options in production environments so only the servlets and folders the developers had intended to are accessible.



web.xml
*ibm-web-ext.xml
Web Extensions Editor - RAD75EJBWeb
Overview
Define the main components of the extensions in this section.
type filter text
Web Extensions
Add...
Remove
Up
Down
Details
Set the properties for the selected item. Required fields are denoted by "*".
Design
Source

Default Error Page URI:	
Reload Interval:	3
Auto Encode Requests:	☐
Auto Encode Responses:	☐
Autoload Filters:	☐
Enable Directory Browsing:	☐
Enable File Serving:	✓
Enable Reloading:	✓
Enable Serving Servlets By Class Names:	☐
Pre-compile JSPs:	☐

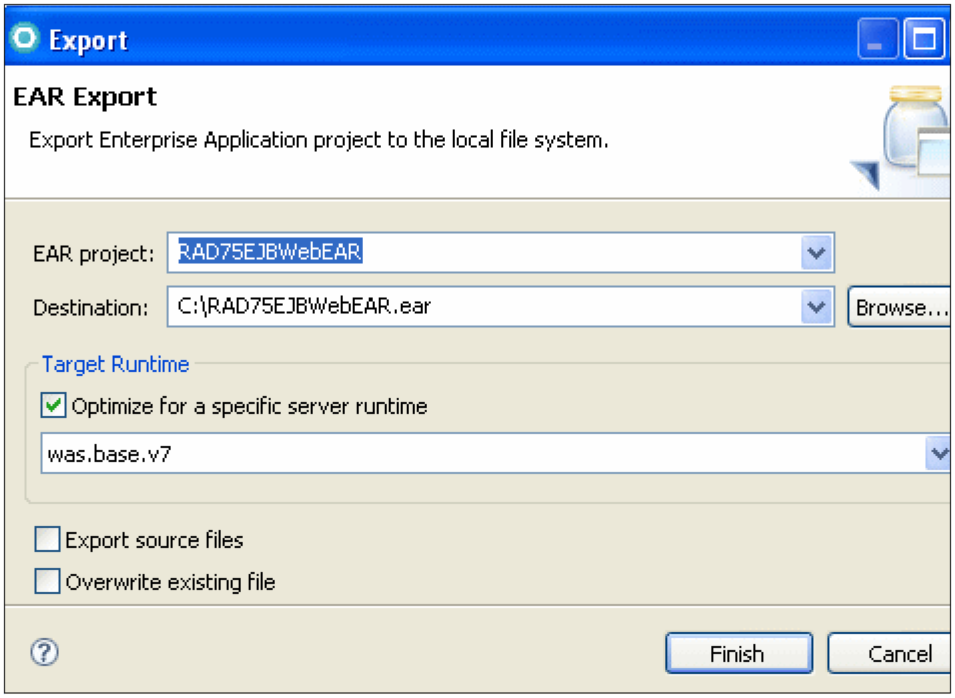
Figure 10 Web module extensions editor

5. When done, press Ctrl-S to save the deployment descriptor.

Exporting to an EAR file

To export the RAD75EJBWebEAR application as an EAR file with all its dependent modules, do the following steps:

1. Select the **RAD75EJBWebEAR** project and right-click. Select **Export...** and **EAR file** from the pop-up menu.
2. In the Export dialog, browse to a destination, such as C:\, as shown in Figure 11.


Export
EAR Export
Export Enterprise Application project to the local file system.

EAR project: RAD75EJBWebEAR

Destination: C:\RAD75EJBWebEAR.ear

Target Runtime

[✓] Optimize for a specific server runtime

was.base.v7

[ ] Export source files

[ ] Overwrite existing file

Figure 11   Exporting Enterprise Project to EAR file

3. Click **Finish**.
4. The EAR file exported is now prepared for installation in WebSphere Application Server v7.0.

You can optionally add more configuration details to the EAR file by creating a WebSphere Enhanced EAR file.

WebSphere Enhanced EAR

A WebSphere enhanced EAR is a regular JEE EAR file, but with additional configuration information for resources required by JEE applications. While adding this extra configuration information at packaging time is not mandatory, it can simplify deployment of JEE applications to WebSphere if the environments where the application is to be deployed are similar.

When an Enhanced EAR is deployed to WebSphere Application Server, WebSphere can automatically configure the resources specified in the Enhanced

EAR. This reduces the number of configuration steps required to set up the WebSphere environment to host the application.

When an Enhanced EAR is uninstalled, the resources that are defined at the application level scope are removed as well. However, resources defined at a scope other than application level are not removed because they might be in use by other applications. Resources created at the Application level scope are limited in visibility to only that application.

Table 6 shows the resources supported by the Enhanced EAR and the scope in which they are created.

Table 6 Scope for resources in WebSphere Enhanced EAR file

Resource	**Scope**
JDBC™ providers	Application
Data sources	Application
Resource adapters	Application
JMS resources	Application
Substitution variables	Application
Class loader policies	Application
Shared libraries	Server
JAAS authentication aliases	Cell
Virtual hosts	Cell

J2C Resource Adapters can be configured either as embedded or external resources. An embedded RAR is packaged within an enterprise application (EAR), deployed as a part of JEE application installation, and is removed when the application is uninstalled from the server. An external RAR is packaged as a standalone RAR file, is deployed explicitly on a WebSphere node, and is not managed as a JEE application. If an adapter is used only by a single application, it should be configured as an embedded RAR. If it is to be shared between multiple applications, it should be an external RAR.

To view the application scoped resources using the administrative console, select **Applications** → **Application Types** → **WebSphere Enterprise Applications** → ***<application>***. Select **Application scoped resources** in the References section. If there are no application scoped resources, you will not see this option.

Configuring a WebSphere Enhanced EAR

The supplemental information in an Enhanced EAR is modified by using the WebSphere Application Server Deployment editor. The information itself lives in XML files in a folder called `ibmconfig` in the EAR file's META-INF folder.

To access the Enhanced EAR deployment options right-click the **RAD75EJBWebEAR** project and select **Java EE**, and then the **Open WebSphere Application Server Deployment** option. This opens up the editor as shown in Figure 12.

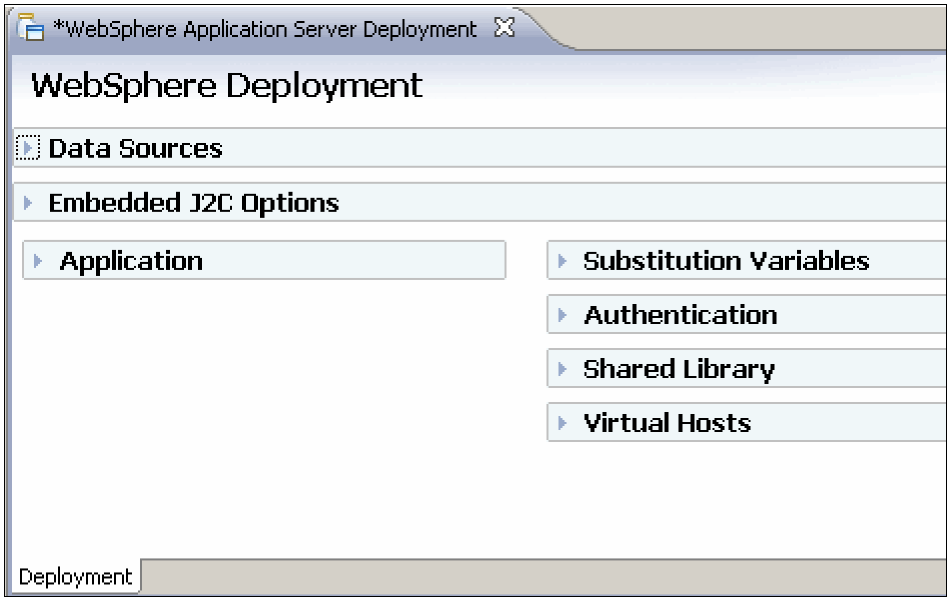
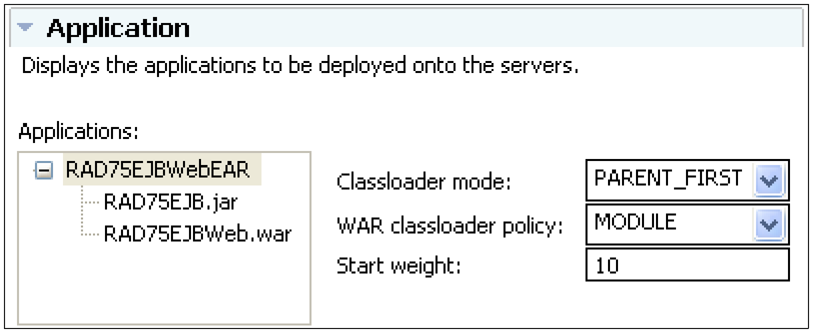


Figure 12 WebSphere Enhanced EAR editor

In the Application section in Figure 13, you can see the class loader policies and class loader mode configured for each of the containing module. ITSO Bank runs fine with the default policies and modes, so they do not need to be changed.


Application

Displays the applications to be deployed onto the servers.

Applications:

- RAD75EJBWebEAR
  - RAD75EJB.jar
  - RAD75EJBWeb.war

Classloader mode: PARENT_FIRST

WAR classloader policy: MODULE

Start weight: 10

Figure 13 Configuring class loader mode and class loader policies

When creating an Enhanced EAR file, RAD-AD automatically adds a JDBC provider for the Derby database. We will change this to use DB2 instead as that is what we will use when deploying the application. To achieve this we need to add the following items:

- JAAS authentication alias
- JDBC provider for DB2
- Data source for DB2 database

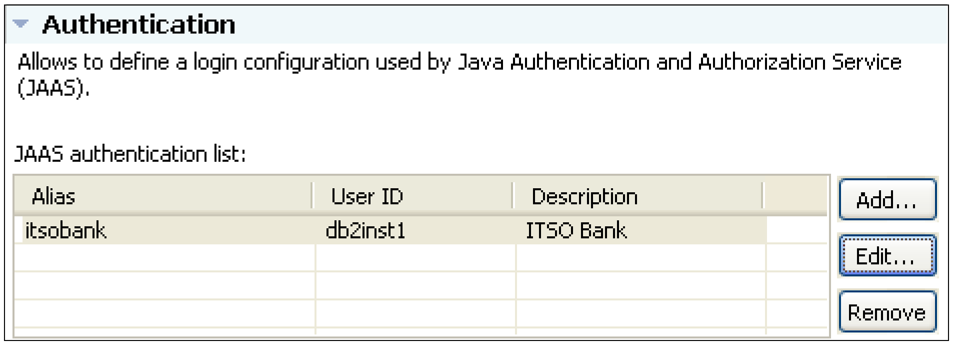
Just to show the use of the editor, we will also configure a new virtual host for a domain called www.itsobank.ibm.com.

Configuring a JAAS authentication alias

To configure the JAAS authentication alias, do the following steps:

1. Expand the **Authentication** section.
2. Click the **Add** button.
3. In the dialog box that displays, enter:
   - itsobank as the alias
   - A user ID with access to the ITSOBANK database (db2inst1 in our case)
   - The password for the user ID.
   - ITSO Bank as the description

4. Click **OK**. See Figure 14 for the results.



Authentication

Allows to define a login configuration used by Java Authentication and Authorization Service (JAAS).

JAAS authentication list:

Alias	User ID	Description
itsobank	db2inst1	ITSO Bank

Figure 14 Configuring JAAS authentication alias for ITSO Bank

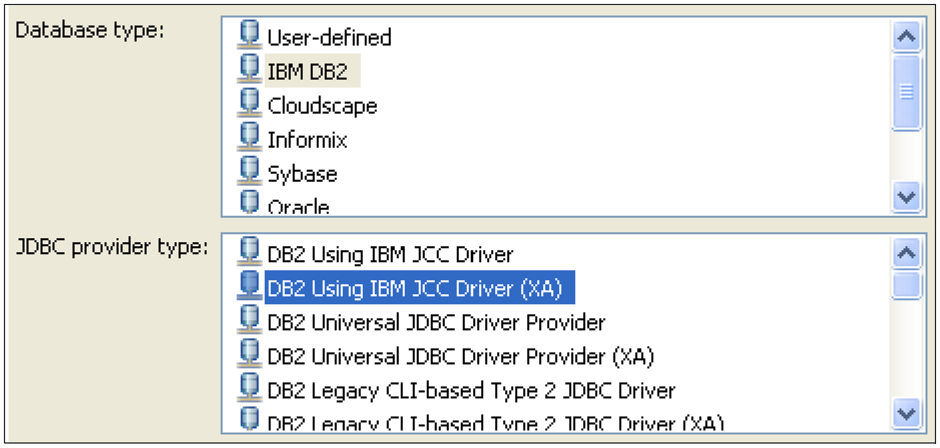
Configuring a DB2 JDBC provider

JDBC providers are configured in the Data Sources section, so expand this section. Before adding the DB2 provider, first delete the pre-configured Derby JDBC Provider (XA) provider by selecting it and clicking the **Remove** button.

To configure the DB2 JDBC provider, do the following steps:

1. Click the **Add** button next to the JDBC provider list.
2. In the dialog box:
   - Select **IBM DB2** as the Database type.
   - Select **DB2 Using IBM JCC Driver (XA)** as the JDBC provider type.

   See Figure 15.



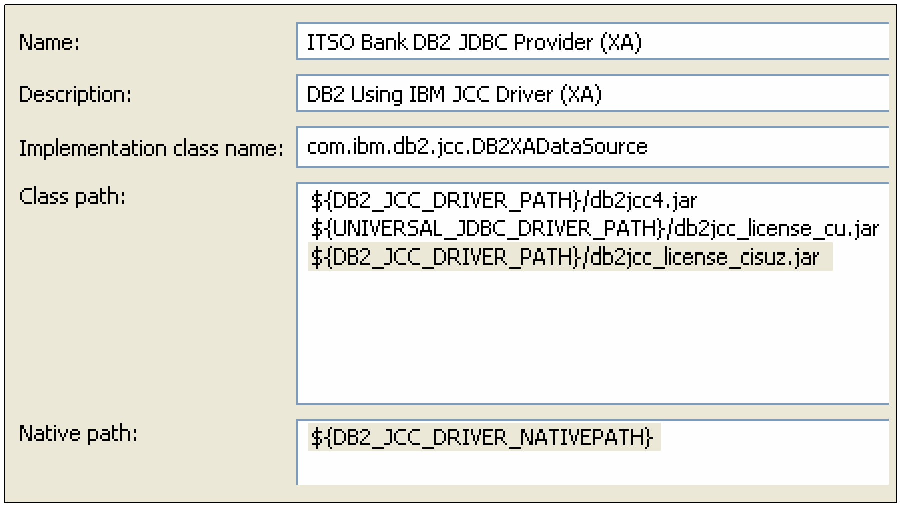
Database type:
- User-defined
- IBM DB2
- Cloudscape
- Informix
- Sybase
- Oracle

JDBC provider type:
- DB2 Using IBM JCC Driver
- DB2 Using IBM JCC Driver (XA)
- DB2 Universal JDBC Driver Provider
- DB2 Universal JDBC Driver Provider (XA)
- DB2 Legacy CLI-based Type 2 JDBC Driver
- DB2 Legacy CLI-based Type 2 JDBC Driver (XA)

Figure 15 Creating a DB2 JDBC Provider

Click **Next**.

3. In the next dialog box, enter a name for the JDBC provider (for administration purposes only) and leave the other properties as the default values. See Figure 16.

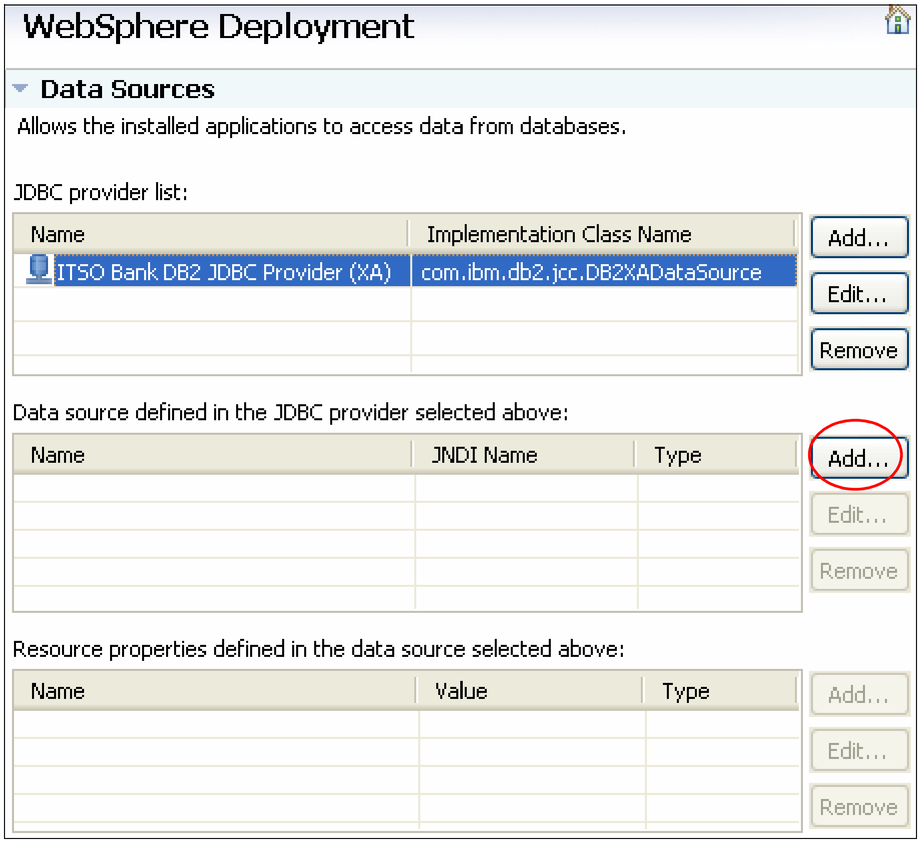


Name:	ITSO Bank DB2 JDBC Provider (XA)
Description:	DB2 Using IBM JCC Driver (XA)
Implementation class name:	com.ibm.db2.jcc.DB2XADataSource
Class path:	${DB2_JCC_DRIVER_PATH}/db2jcc4.jar ${UNIVERSAL_JDBC_DRIVER_PATH}/db2jcc_license_cu.jar ${DB2_JCC_DRIVER_PATH}/db2jcc_license_cisuz.jar
Native path:	${DB2_JCC_DRIVER_NATIVEPATH}

Figure 16 Creating a DB2 JDBC provider

Click **Finish**.

4. Select the **ITSO Bank DB2 JDBC Provider (XA)** you just created and click the **Add** button next to the Data source list, as in Figure 17.


WebSphere Deployment

Data Sources

Allows the installed applications to access data from databases.

JDBC provider list:

Name	Implementation Class Name
ITSO Bank DB2 JDBC Provider (XA)	com.ibm.db2.jcc.DB2XADataSource

Data source defined in the JDBC provider selected above:

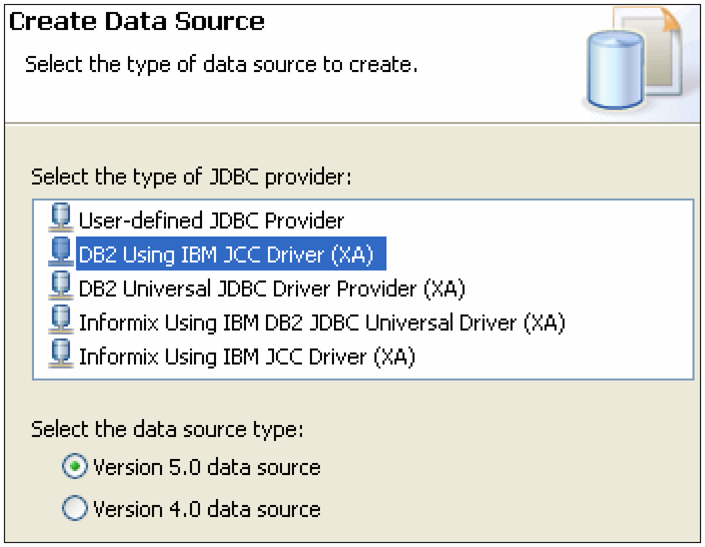
Name	JNDI Name	Type

Resource properties defined in the data source selected above:

Name	Value	Type

Figure 17 Creating a DB2 data source

5. In the Create a Data Source dialog box, select **DB2 Using IBM JCC Driver (XA)** as the JDBC provider type and **Version 5.0 data source** as the data source type, as in Figure 18.


Create Data Source

Select the type of data source to create.

Select the type of JDBC provider:

- User-defined JDBC Provider
- DB2 Using IBM JCC Driver (XA)
- DB2 Universal JDBC Driver Provider (XA)
- Informix Using IBM DB2 JDBC Universal Driver (XA)
- Informix Using IBM JCC Driver (XA)

Select the data source type:

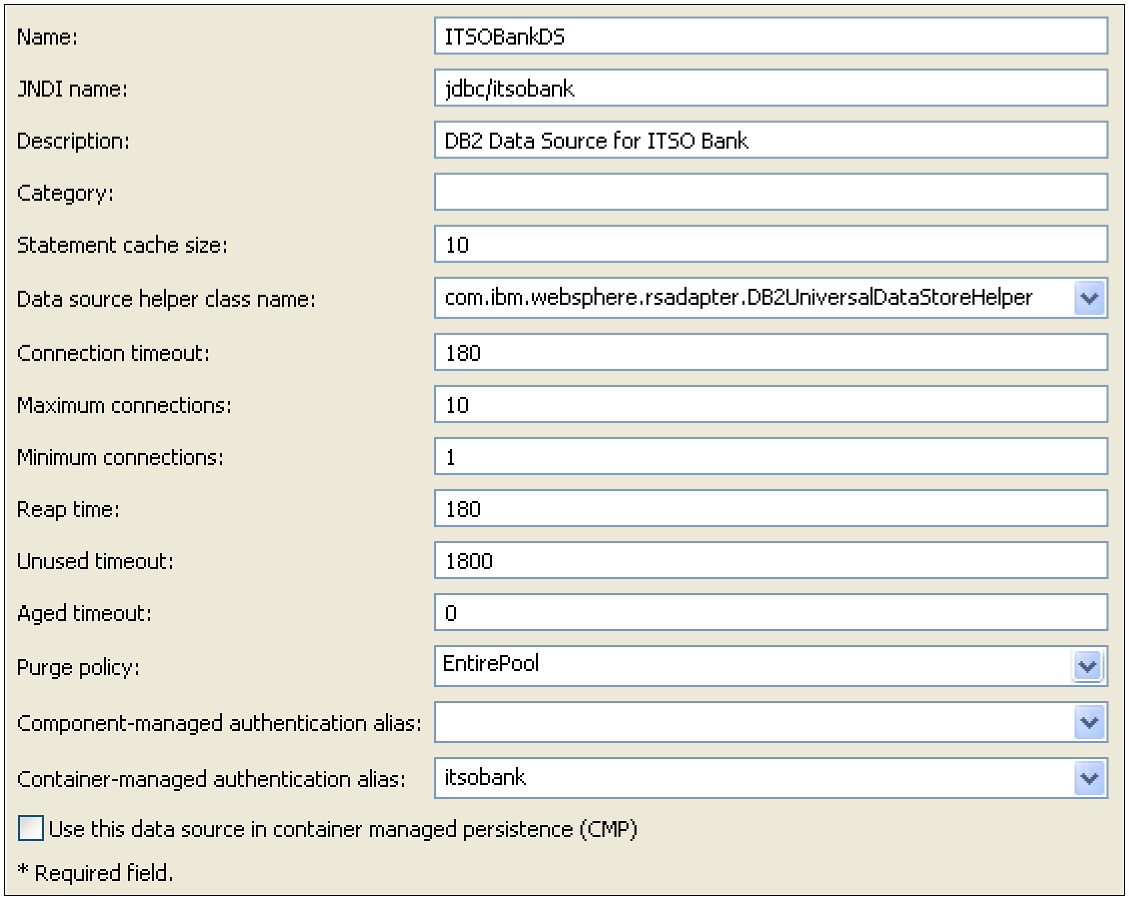
[✓ Version 5.0 data source]

Version 4.0 data source

Figure 18 Creating a DB2 data source

Click **Next**.

6. In the dialog box displayed enter the appropriate values for the DB2 data source. See Figure 19.



Name:	ITSOBankDS
JNDI name:	jdbc/itsobank
Description:	DB2 Data Source for ITSO Bank
Category:	
Statement cache size:	10
Data source helper class name:	com.ibm.websphere.rsadapter.DB2UniversalDataStoreHelper
Connection timeout:	180
Maximum connections:	10
Minimum connections:	1
Reap time:	180
Unused timeout:	1800
Aged timeout:	0
Purge policy:	EntirePool
Component-managed authentication alias:	
Container-managed authentication alias:	itsobank

☐ Use this data source in container managed persistence (CMP)

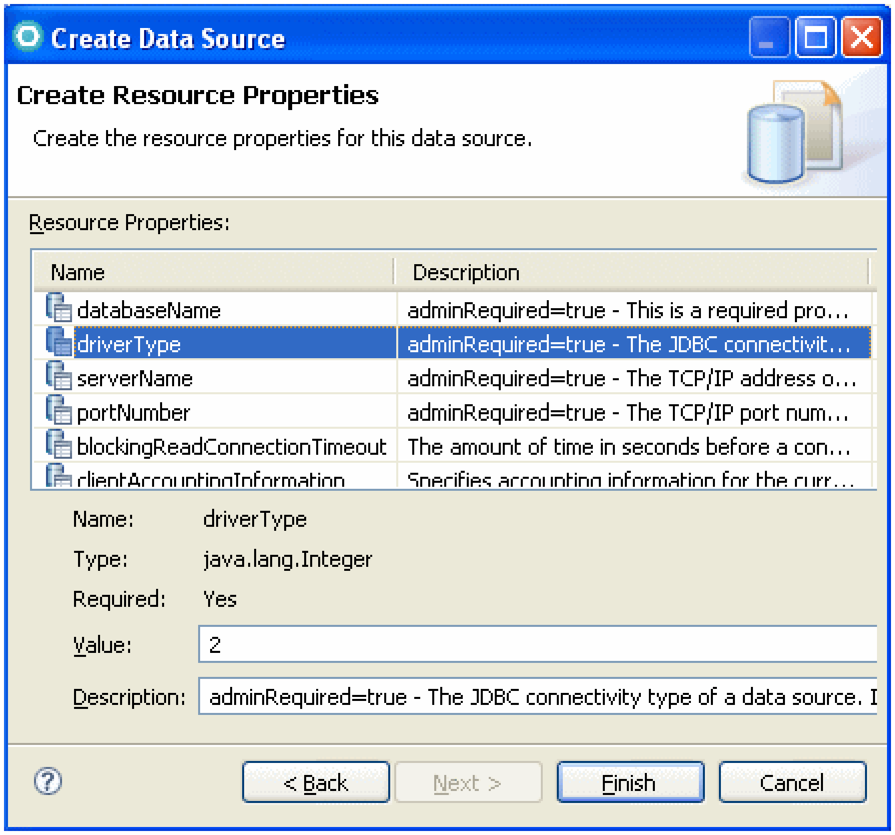
* Required field.

Figure 19 Creating a DB2 data source

- Enter `ITSOBankDS` as the name.
- Enter `jdbc/itsobank` as the JNDI name.
- Enter `DB2 Data Source` for ITSO Bank as the description.
- Select **itsobank** as the Container-managed authentication alias (you might need to scroll the window to the far right to see the pull down arrow).
- Uncheck **Use this data source in container manager persistence (CMP)**. The ITSO Bank application uses JPA for persistence so we do not need to add support for CMP Entity beans for this data source.

Click **Next**.

7. Highlight the **databaseName** property in the Resource properties section. Enter ITSOBANK in the Value field. Then highlight the **driverType** property and change the value from type 4 to type 2. Type 2 means that the DB2 database is installed on the same machine as WebSphere Application Server, or that the DB2 Client software is installed on the same machine and configured to handle the remote connection if the database is on a remote machine. A type 4 driver can connect directly to a remote database over TCP/IP. See Figure 20.



Create Resource Properties

Create the resource properties for this data source.

Resource Properties:

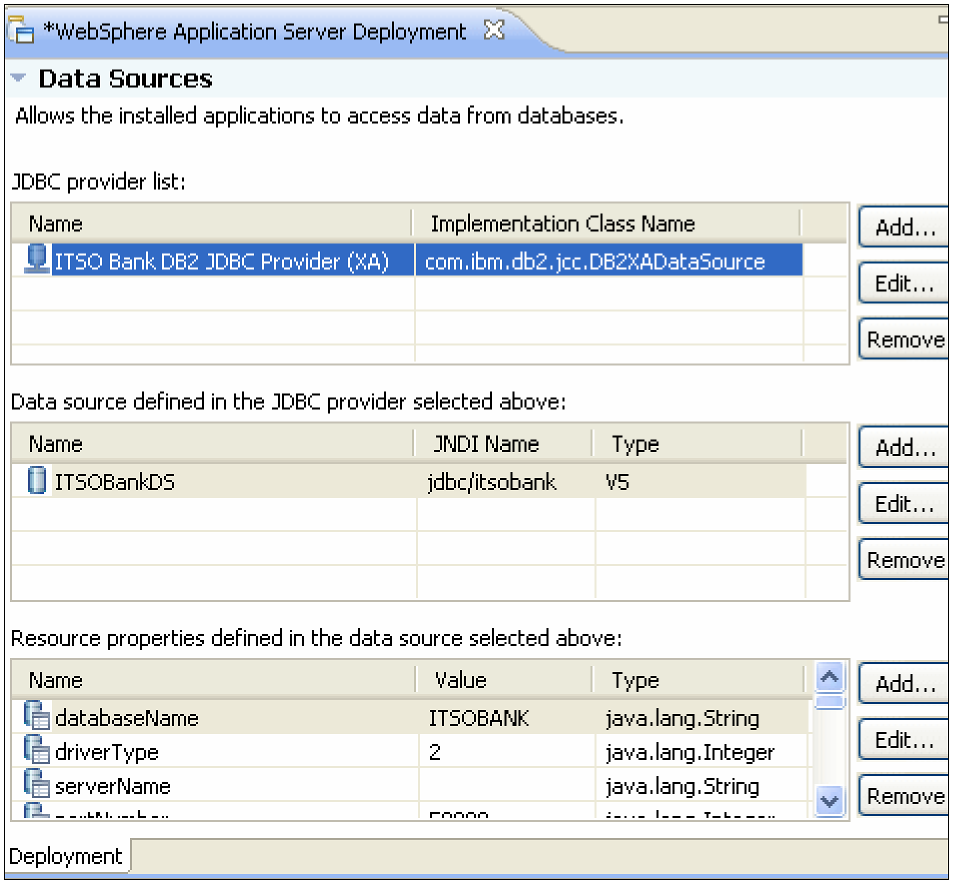
Name	Description
databaseName	adminRequired=true - This is a required pro...
driverType	adminRequired=true - The JDBC connectivit...
serverName	adminRequired=true - The TCP/IP address o...
portNumber	adminRequired=true - The TCP/IP port num...
blockingReadConnectionTimeout	The amount of time in seconds before a con...
clientAccountingInformation	Specifies accounting information for the curr...

Name:	driverType
Type:	java.lang.Integer
Required:	Yes
Value:	2
Description:	adminRequired=true - The JDBC connectivity type of a data source. I

Figure 20 Setting database name and driver type

Click **Finish**.

When you are finished, your data source configuration should look like Figure 21.


*WebSphere Application Server Deployment

Data Sources

Allows the installed applications to access data from databases.

JDBC provider list:

Name	Implementation Class Name
ITSO Bank DB2 JDBC Provider (XA)	com.ibm.db2.jcc.DB2XADataSource

Data source defined in the JDBC provider selected above:

Name	JNDI Name	Type
ITSOBankDS	jdbc/itsobank	V5

Resource properties defined in the data source selected above:

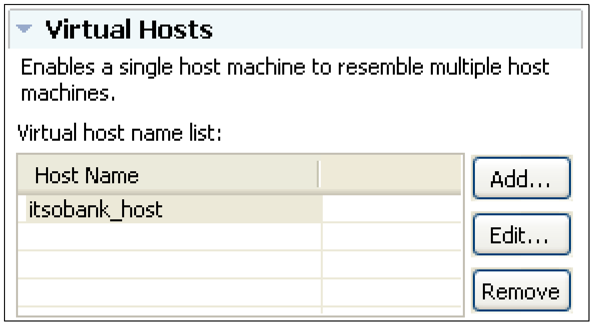
Name	Value	Type
databaseName	ITSOBANK	java.lang.String
driverType	2	java.lang.Integer
serverName		java.lang.String

Figure 21   DB2 data source configured

Adding a virtual host

To configure a virtual host, do the following steps:

1. Expand the **Virtual Hosts** section of the Deployment tab and click the **Add** button next to the Virtual host name list.
2. In the Add Host Name Entry dialog box, enter itsobank_host and click **OK**. Your new virtual host will appear in the Virtual Hosts list. See Figure 22.



Virtual Hosts

Enables a single host machine to resemble multiple host machines.

Virtual host name list:

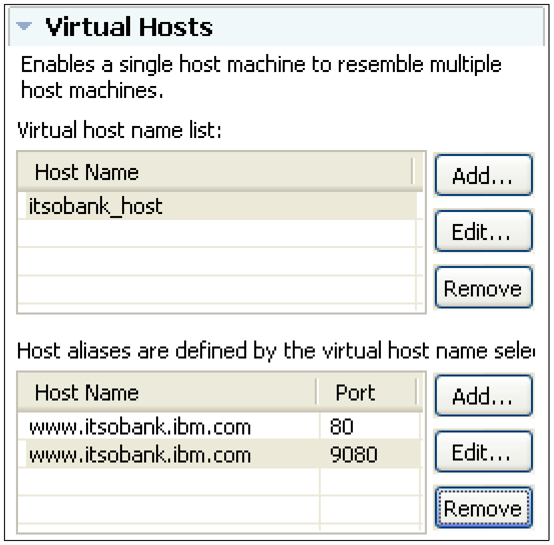
Host Name
itsobank_host

Figure 22 Add a new virtual host

3. Click the **Add** button next to the Host aliases list.
4. In the Add Host Alias Entry dialog box, enter www.itsobank.ibm.com for the host name and 80 for the port number. Click **OK**.

   Repeat the procedure and add number 9080 as well. We will use this port when we deploy the application later. If your server uses another port use that port number instead. You can have as many host aliases as you like.

   The host aliases will appear in the list (Figure 23).



Virtual Hosts

Enables a single host machine to resemble multiple host machines.

Virtual host name list:

Host Name
itsobank_host

Host aliases are defined by the virtual host name sele

Host Name	Port
www.itsobank.ibm.com	80
www.itsobank.ibm.com	9080

Figure 23 Configuring the virtual host for ITSO Bank

5. When you are finished, press **Ctrl-S** to save the deployment descriptor editor.

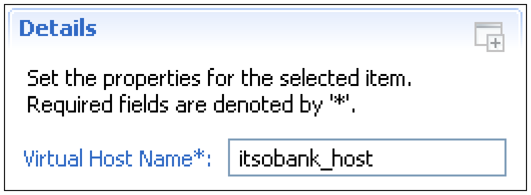
Setting the default virtual host for Web modules

Just because we have configured a new virtual host, itsobank_host, in the Enhanced EAR file does not mean that all our Web modules automatically use it.

The default virtual host for a Web module created in the RAD or RAD-AD is default_host, which is also the case for the ITSO Bank application.

To modify the Web modules to use the itsobank_host instead, do the following steps:

1. Expand the **RAD75EJBWeb** project and double-click the **RAD75EJBWeb** heading as shown in Figure 8 on page 21 to open the Web module deployment descriptor.
2. In the lower right corner of the panel click the **Open WebSphere Bindings** link.
3. Change the Virtual Host Name to itsobank_host, as shown in Figure 24.



Details

Set the properties for the selected item.
Required fields are denoted by "*".

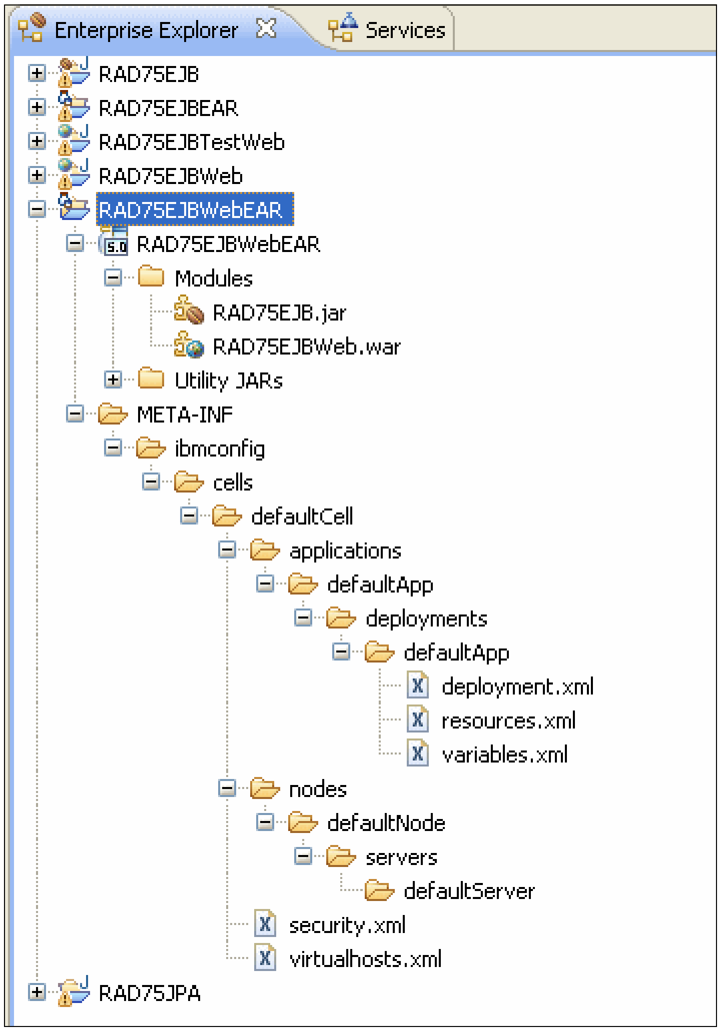
Virtual Host Name*: itsobank_host

Figure 24 Setting default virtual host for a Web module

4. Save the deployment descriptor by pressing **Ctrl-S** and then close it.

Examining the WebSphere Enhanced EAR file

The information about the resources configured is stored in the ibmconfig subdirectory of the EAR file's META-INF directory. Expanding this directory reveals the well-known directory structure for a cell configuration, as seen in Figure 25. You can also see the scope level where each resource is configured.



```
RAD75EJB
RAD75EJBEAR
RAD75EJBTestWeb
RAD75EJBWeb
RAD75EJBWebEAR
    RAD75EJBWebEAR
        Modules
            RAD75EJB.jar
            RAD75EJBWeb.war
        Utility JARs
    META-INF
        ibmconfig
            cells
                defaultCell
                    applications
                        defaultApp
                            deployments
                                defaultApp
                                    deployment.xml
                                    resources.xml
                                    variables.xml
                    nodes
                        defaultNode
                            servers
                                defaultServer
                    security.xml
                    virtualhosts.xml
RAD75JPA
```

Figure 25 Enhanced EAR file contents

At deployment time, WebSphere Application Server uses this information to automatically create the resources.

After you have added the additional configuration information to the application you should now export the project as an EAR file. Refer to “Exporting to an EAR file” on page 22 for information about how to do this.

Packaging recommendations

Here are some basic rules to consider when packaging an enterprise application:

- EJB JAR modules and Web WAR modules comprising an application should be packaged together in the same EAR module, and executed within the same application server JVM process. This is to avoid remote EJB calls (RMI/IIOP) across application server JVM processes, which is costly from a performance perspective.
- Utility classes used by a single Web module should only be placed within its WEB-INF/lib folder.
- Utility classes used by multiple modules within an enterprise application should be placed at the root of the EAR file as Utility Projects, so they are accessible both by servlets and EJBs.
- Utility classes used by multiple enterprise applications can be placed outside the applications on a directory referenced through a shared library definition.

See Chapter 13, *Understanding class loaders* for more details on how WebSphere finds and loads classes.

Business-level applications

A business-level application is a concept which aims to expand the notion of "Application" beyond JEE. Its administration model provides the entire definition of an application as it makes sense to the business. In contrast with an enterprise application (EAR file) a business-level application is only a logical WebSphere configuration artifact, similar to a server or cluster, that is stored in the configuration repository.

Figure 26 shows the concept of business-level applications.

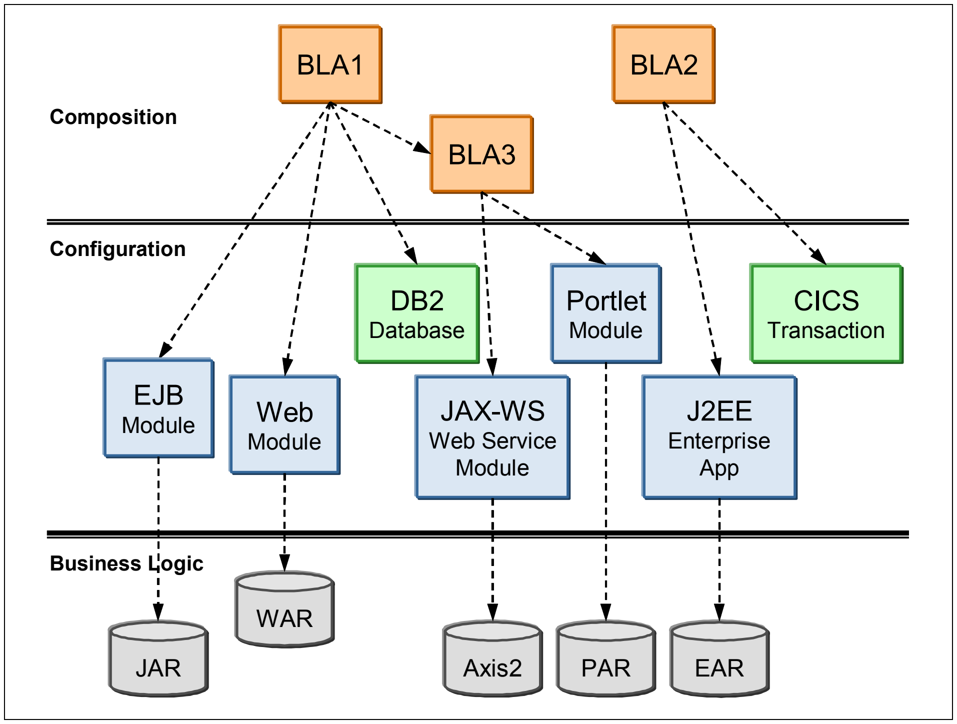


Figure 26 Business-level application concept

Business-level applications can be used in several different ways. Often a business application, such as an Order System, does not consist of only one enterprise application (EAR), but rather multiple applications that must all be running for the whole business application to work.

One way of using business-level applications is to group the separate enterprise applications which make up the business application into one manageable unit that can be started, stopped, updated, and so on. But a business-level application cannot only reference JEE components, but also assets that are not part of the JEE concept. For example, CORBA (C++) executables hosted in a Generic Server or files on the file system that are not managed by WebSphere but still required by the application.

A business-level application does not represent or contain application binary files, however. Instead it is a configuration that lists one or more composition units that represent the application binary files. A business-level application uses the binary files to run the application business logic. Administration of binary files is done using the normal methods for managing modules (for example, Web or EJB modules) and is separate from administration of the application definition.

A business-level application does not introduce any new programming, runtime, or packaging models:

- You do not need to change your application business logic. The business-level application function does not introduce new application programming interfaces (APIs).
- You do not need to change your application runtime settings. WebSphere supports all of the runtime characteristics, such as security, class loading and isolation, required by individual programming models to which business components are written.
- You do not need to change your application packaging. There is no specific unique packaging model that provides a business-level application definition.

Note: Business-level applications are only supported on WebSphere Application Server v7.0 nodes. They are not supported on any previous versions of WebSphere Application Server.

The terminology for business-level applications introduces two new terms; *assets* and *composition units*.

An *asset* represents one or more application binary files that are stored in an asset repository. Typical assets include application business logic such as EAR files, EJB modules, Web modules, Service Component Architecture (SCA) modules, shared library files, static content, and other resource files. The asset repository is managed by WebSphere Application Server and does not require any 3rd party software.

You must register files as assets before you can add them to one or more business-level applications. At the time of asset registration, you can import the physical application files into WebSphere’s configuration repository or you can specify an external location where the asset resides.

A *composition unit* represents a configured asset in a business-level application. Configured in this context means installed, so a configured Web module means a Web module which is installed.

WebSphere Application Server handles three types of composition units:

- Asset composition units:

  Composition units created from assets by configuring each deployable unit of the asset to run on deployment targets.

- Shared library composition units:

  Composition units created from JAR-based assets by ignoring all the deployable objects from the asset and treating the asset JAR file as a library of classes.
- Business-level application composition units:

  Composition units created from business-level applications that are added to existing business-level applications.

Figure 27 shows the relationship between assets, composition units, and business-level applications.

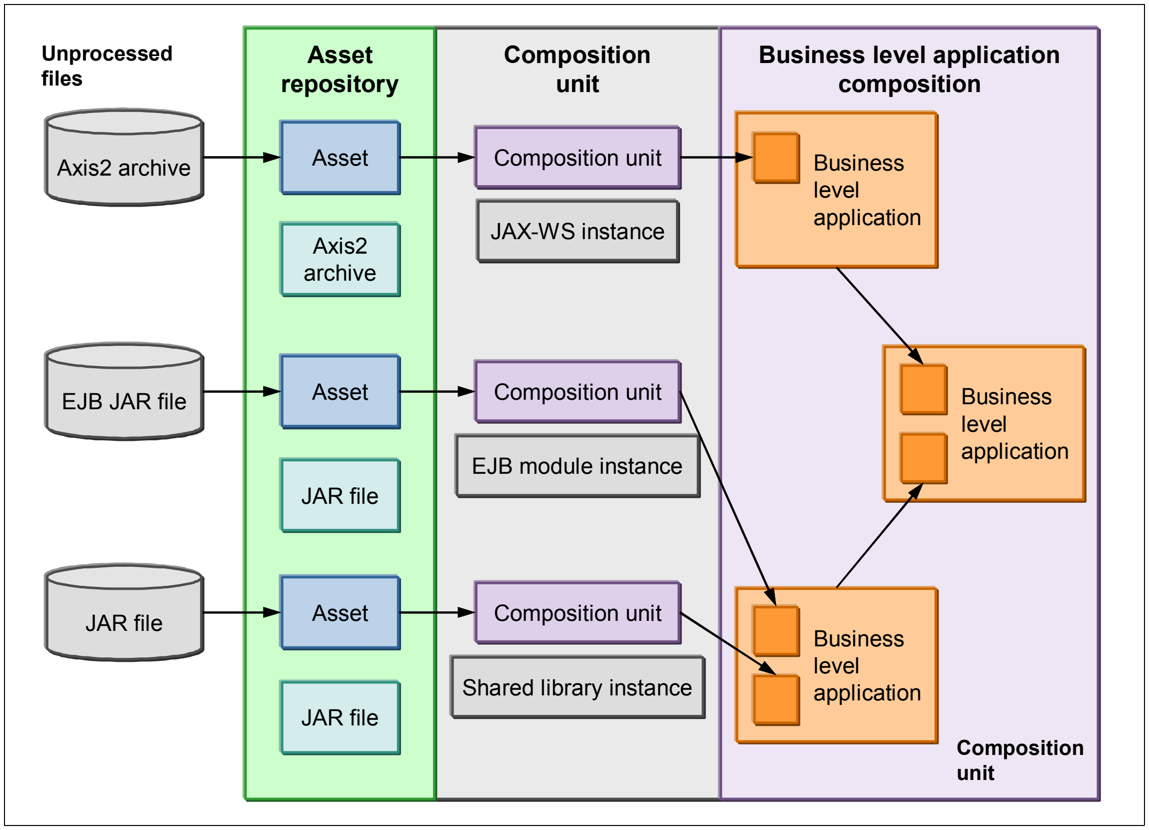


Figure 27   Relationship between business-level application artifacts

If an asset depends on another asset, a relationship can be created between them. For example, this would allow a Web module asset to reference classes in a shared library.

So, to summarize, a business-level application consists of composition units. When you add an asset to a business-level application, a composition unit is created for the asset. The composition unit is the configured (installed) asset.

Example: Creating a business-level application

As a basic example of how to create a business-level application, we use the RAD75EJBWeb application (see “Example: Packaging an application” on page 17) and combine it with the WebSphere DefaultApplication’s Web module (which includes, for example, the SnoopServlet) which can be extracted from the DefaultApplication’s EAR file. We create one asset for the RAD75EJBWeb application and another for the DefaultWebApplication. We then create a business-level application containing these two assets.

To create the two assets, do the following steps:

1. Open the WebSphere administrative console and select **Applications** → **Application Types**. Click the **Assets** link.
2. Click the **Import** button. Check the **Local file system** box and click the **Browse** button to locate the RAD75EJBWebEAR.ear file. Select the file and click **Open**. Then click the **Next** button.
3. Step1: Select options for importing an asset:

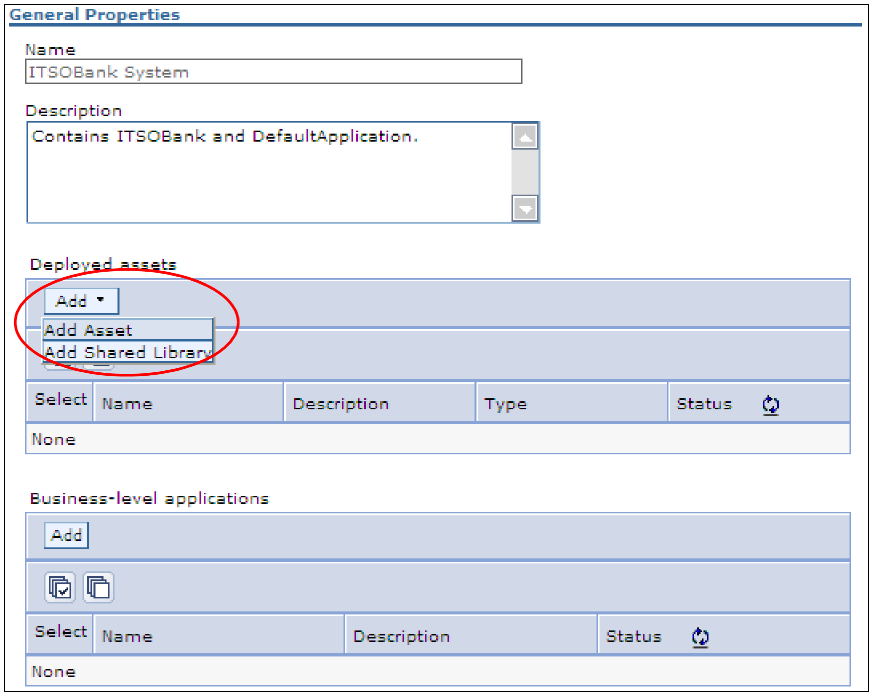
   Enter a brief description of the asset, if wanted. If you want to import the asset to a specific path on your file system enter the path in the Asset binaries destination URL field. If you leave this field blank, the file will be imported to its default location, which is *profile_root*/installedAssets/asset_name/BASE/. Click **Next**.

   Tip: If specifying another name for the asset, you must make sure to keep the asset’s file extension (such as .ear or .war), otherwise WebSphere cannot keep track of the asset type, and fails to import the asset.

4. On the Summary page, click **Finish**. The asset is now imported into WebSphere’s asset repository, but its not yet configured (so it is still just an asset, not a composition unit).
5. Repeat the process and import the DefaultWebApplication.war file. Because the two assets do not depend on each other and do not require any shared library, you do not need to set up any relationships.
6. **Save to master configuration** when done.

   When the assets have been imported into the asset repository, we can now create a business-level application.

7. Select **Applications** → **Application Types** → **Business-level applications**. Click the **New** button.
8. Enter a name, such as ITSOBank System, and click **Apply**.
9. On the Business-level applications page, click the **Add** button under the Deployed assets section and select the **Add Asset** option, as shown in Figure 28.



General Properties

Name: ITSOBank System

Description: Contains ITSOBank and DefaultApplication.

Deployed assets

Select	Name	Description	Type	Status
None				

Business-level applications

Select	Name	Description	Status
None			

Figure 28   Business-level application configuration panel

10. On the next panel, select the **RAD75EJBWebEAR.ear** asset and click **Continue**.
11. You can now configure (install) this asset with the necessary deployment options. The installation process follows the normal steps for installing an application to WebSphere Application Server.

    For detailed step-by-step instructions on how to do this, refer to Chapter 15: *Deploying applications*. Proceed through the installation panels until the Summary page is shown, and click **Finish**. WebSphere now installs the application, and it is now a composition unit, an asset that has been configured.

Note: At Step 1: Select installation options, WebSphere generates a unique application name such as app1143018803114601914 for the application. You might want to change this to something more descriptive.

12. Select **Applications** → **Application Types** → **Business-level applications** and click the link for the ITSOBank System application.
13. Repeat steps 9 and 10 and add the DefaultWebApplication.war file to the ITSOBank System business-level application as well.
14. **Save** to the master configuration when done.

The business-level application can now be started and stopped by selecting **Applications** → **Application Types** → **Business-level applications** and then clicking the corresponding links.

Note: To delete a business-level application, you must first unmap (delete) the composition units (configured assets) which belong to the business-level application. To do this, select the business-level application, and check the boxes next to the deployed assets and click the **Delete** button. When all assets have been deleted from the business-level application, you can delete the business-level application itself. The assets, however, still remain in WebSphere's asset repository and can be used to configure other business-level applications.

The individual applications, the composition units, which make up the business-level application can be managed individually by selecting **Applications** → **Application Types** → **WebSphere enterprise applications**. and using the links to start, stop, update, and so on.

Notices

This document REDP-4582-00 was created or updated on October 13, 2009.

Send us your comments in one of the following ways:
- Use the online **Contact us** review Redbooks form found at:
  ibm.com/redbooks
- Send your comments in an email to:
  redbook@us.ibm.com
- Mail your comments to:
  IBM Corporation, International Technical Support Organization
  Dept. HYTD Mail Station P099, 2455 South Road
  Poughkeepsie, NY 12601-5400 U.S.A.

Redpaper™

Trademarks