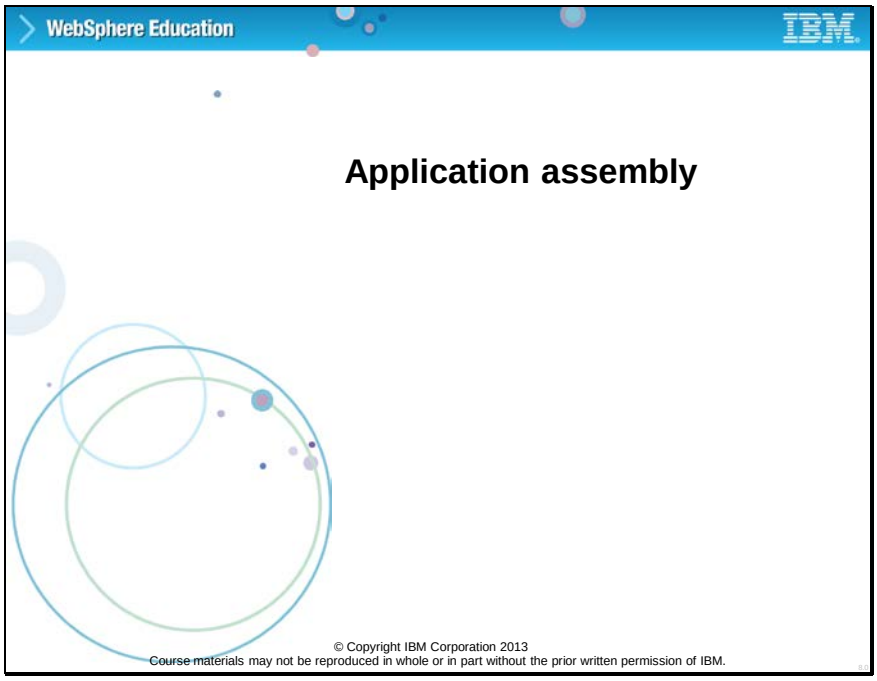
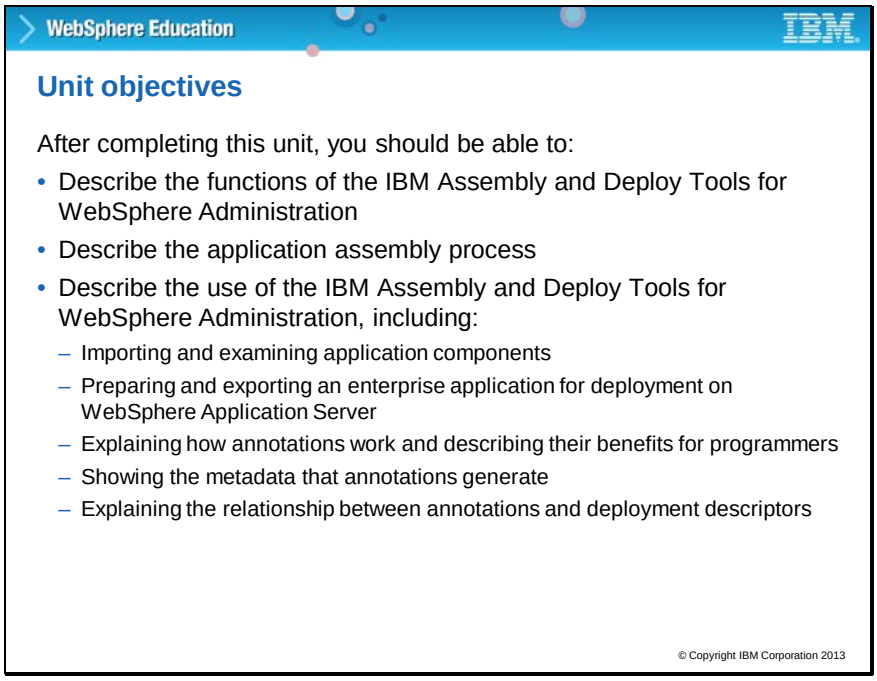
Slide 1



Unit 9: Application assembly

This unit describes the application assembly process and the IBM Assembly and Deploy Tools (IADT) that are used to prepare and export an enterprise application for deployment to WebSphere Application Server.

Slide 2

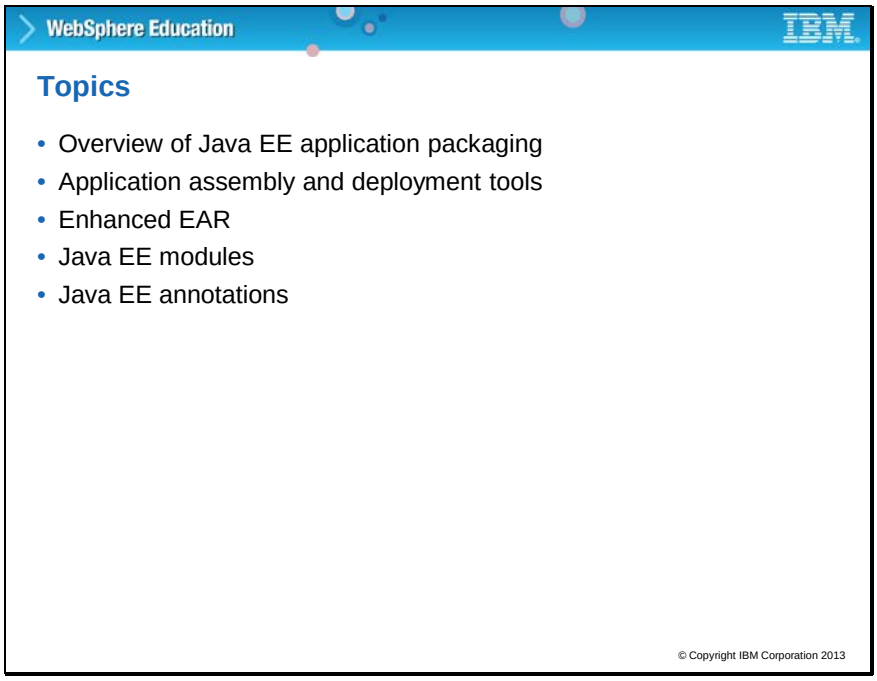


Title: Unit objectives

After completing this unit, you should be able to:

- Describe the functions of the IBM Assembly and Deploy Tools for WebSphere Administration
- Describe the application assembly process
- Describe the use of the IBM Assembly and Deploy Tools for WebSphere Administration, including:
  - Using the IBM Assembly and Deploy Tools for WebSphere Administration to import and examine application components
  - Preparing and exporting an enterprise application for deployment on WebSphere Application Server
  - Explain how annotations work and describe their benefits for programmers
  - Show the metadata that annotations generate
  - Explain the relationship between annotations and deployment descriptors

Slide 3

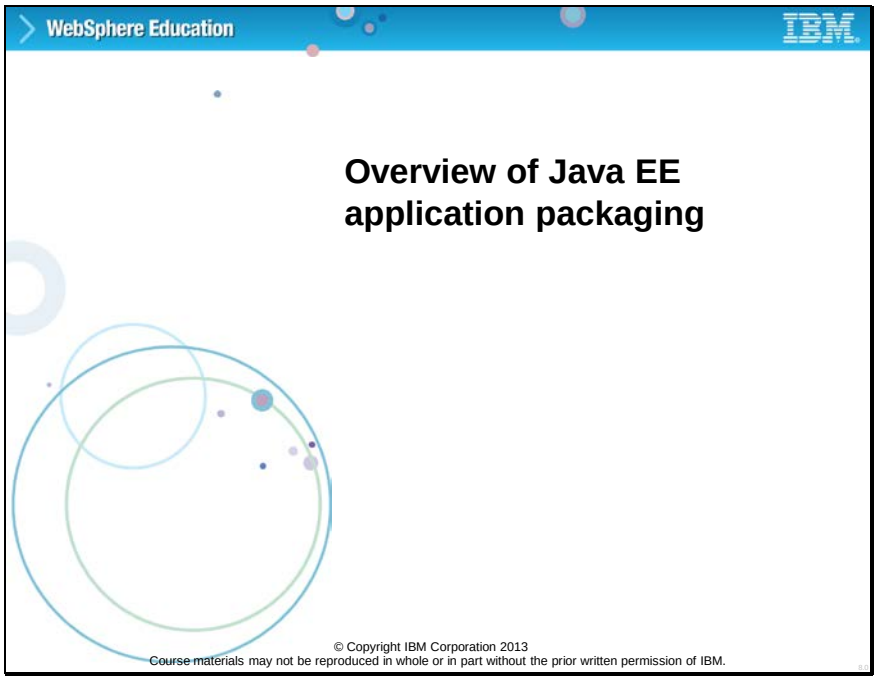


Title: Topics

This unit covers the following topics:

- Overview of Java EE application packaging
- Application assembly and deployment tools
- Enhanced EAR
- Java EE modules
- Java EE annotations

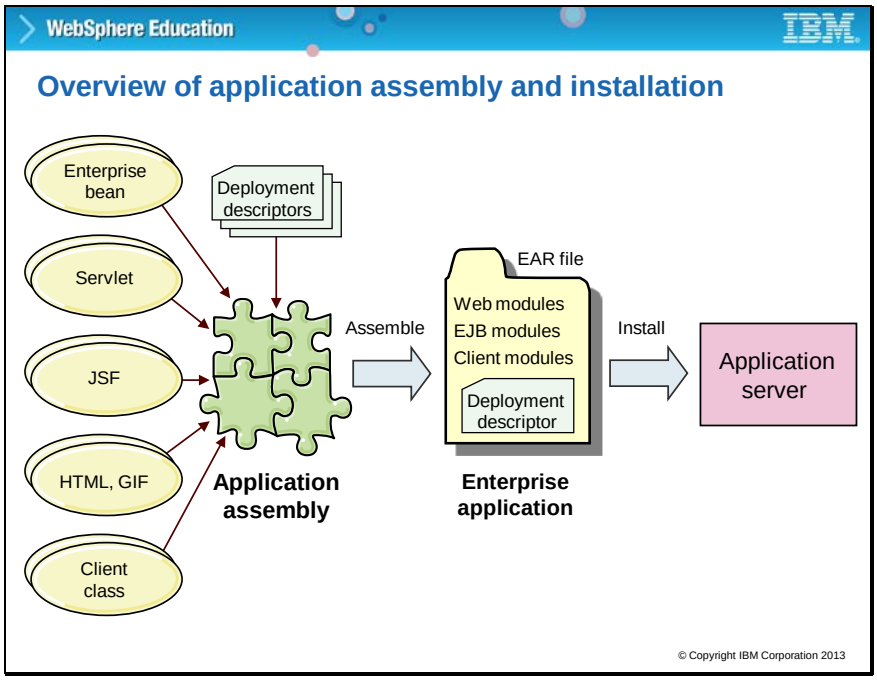
Slide 4



Topic: Overview of Java EE application packaging

This topic provides an overview of Java EE application components and how they are packaged.

Slide 5

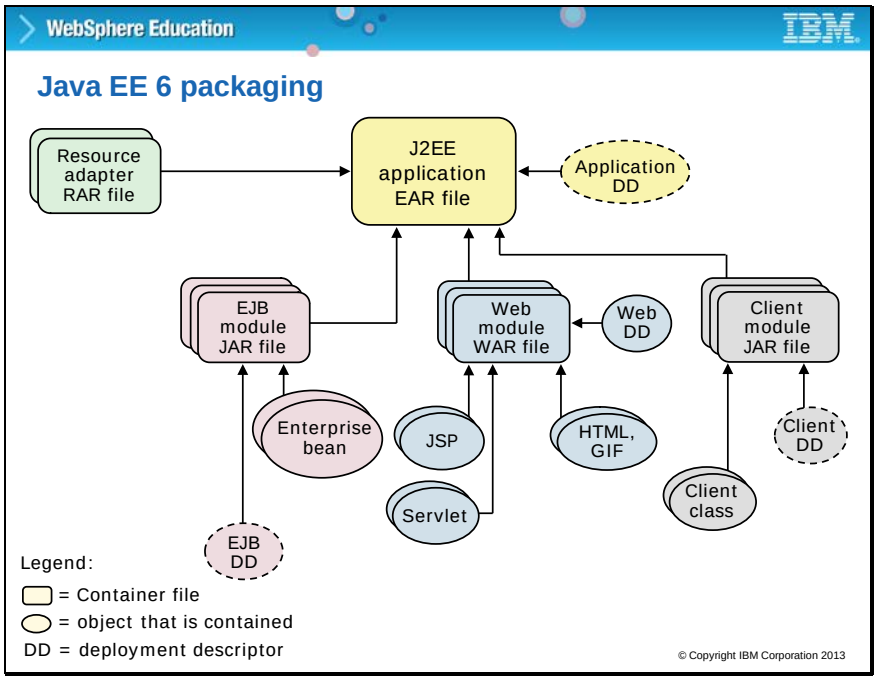


Title: Overview of application assembly and installation

Two main activities are shown in this figure: assembly and installation. This unit describes the assembly process; a following unit describes the installation activity.

This diagram provides a simple overview of the application assembly process. When you assemble code artifacts, you package and configure the code artifacts into deployable Java EE applications and modules, edit annotations or deployment descriptors, and map databases as needed. After assembling your applications, you can deploy the EAR or WAR files onto the application server.

Slide 6



Title: Java EE 6 packaging

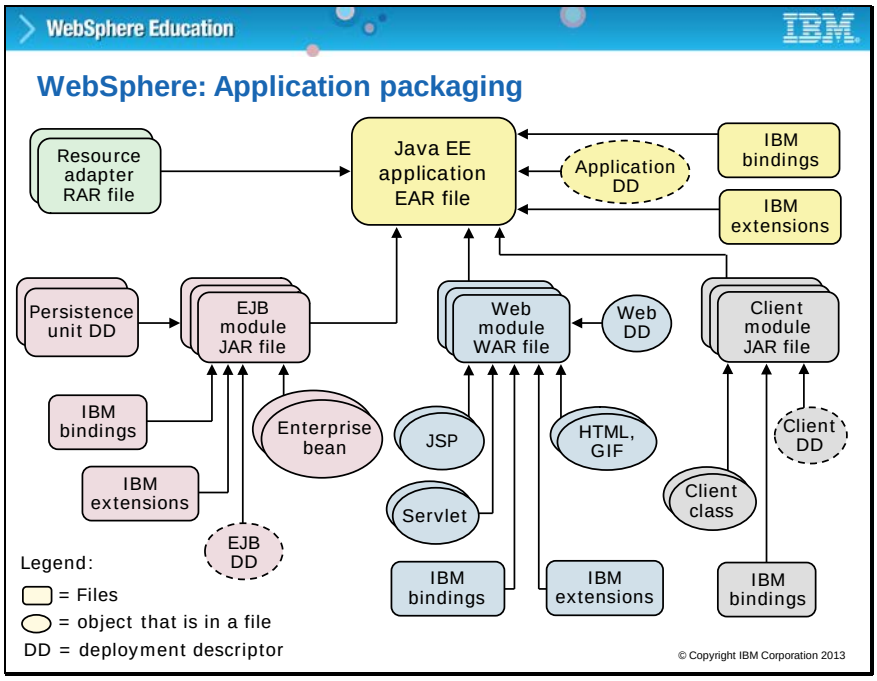
This diagram shows the pieces that can be put together to create a Java EE 6 application.

Enterprise applications can contain the following items:

- EJB modules (packaged in JAR files)
- Web modules (packaged in WAR files)
- Connector modules (packaged in RAR files)
- Session Initiation Protocol (SIP) modules (packaged in SAR files)
- Application client modules
- Other JAR files that contain dependent classes or other components that the application requires
- Any combination of these types of items

A Java EE application is packaged in an enterprise archive (or EAR) file.

Slide 7



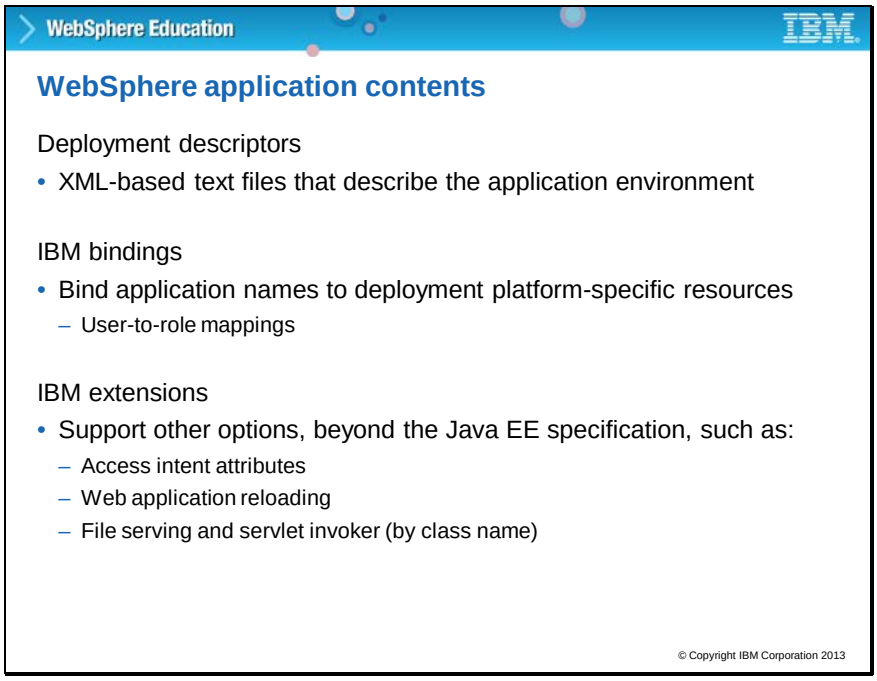
Title: WebSphere: Application packaging

Observe how this diagram adds more items to the enterprise application. Compare it to the previous diagram. These additional items are required for the WebSphere Application Server to "understand" how this application is going to run.

In WebSphere Application Server, the Java EE 6 application EAR file is enhanced with IBM bindings and extensions. Bindings and extensions adapt the generic Java EE application to the IBM WebSphere Application Server environment. Most Java EE server vendors have their own proprietary extensions to the specification. Persistent unit deployment descriptors can be packaged as part of a WAR or EJB JAR file, or they can be packaged as a JAR file. The JAR file can then be included in a WAR or EAR file.

The assembly and deployment tool packages all of these objects in the application EAR file.

Slide 8



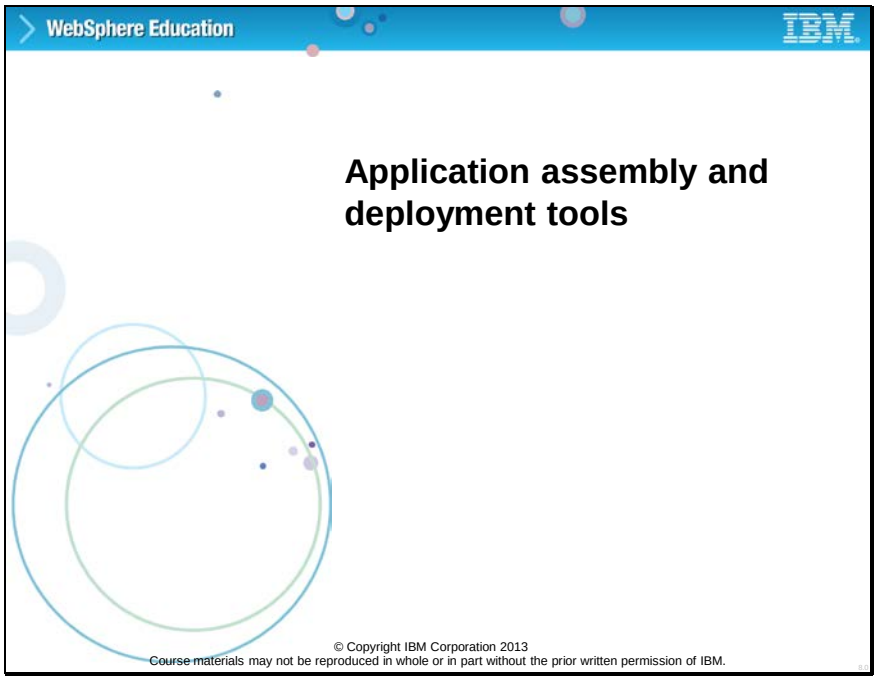
Title: WebSphere application contents

WebSphere applications can contain Java EE standard deployment descriptors, which are XML files that describe how to deploy a module or application by specifying configuration and container options. Features that can be configured at deployment time by using deployment descriptors include security checks and transaction control. IBM bindings associate local names to deployment platform-specific resources; for example, security roles that are mapped to specific authentication credentials. Applications targeting the WebSphere run time can contain more information beyond the Java EE requirements. IBM extensions support more options such as access intent (caching and transaction isolation) and web application reloading. You can specify whether to use WebSphere internal servlets like the file-serving enabler or the invoker, which serves up servlets by class name.

Before an application can start, all enterprise bean (EJB) references and resource references that are defined in the application must be bound to the actual artifacts (enterprise beans or resources) defined in the application server.

When defining bindings, you specify Java Naming and Directory Interface (JNDI) names for the referenceable and referenced artifacts in an application. The jndiName values that are specified for artifacts must be qualified lookup names. An example referenceable artifact is an EJB defined in an application. An example of a referenced artifact is an EJB or a resource reference that the application uses.

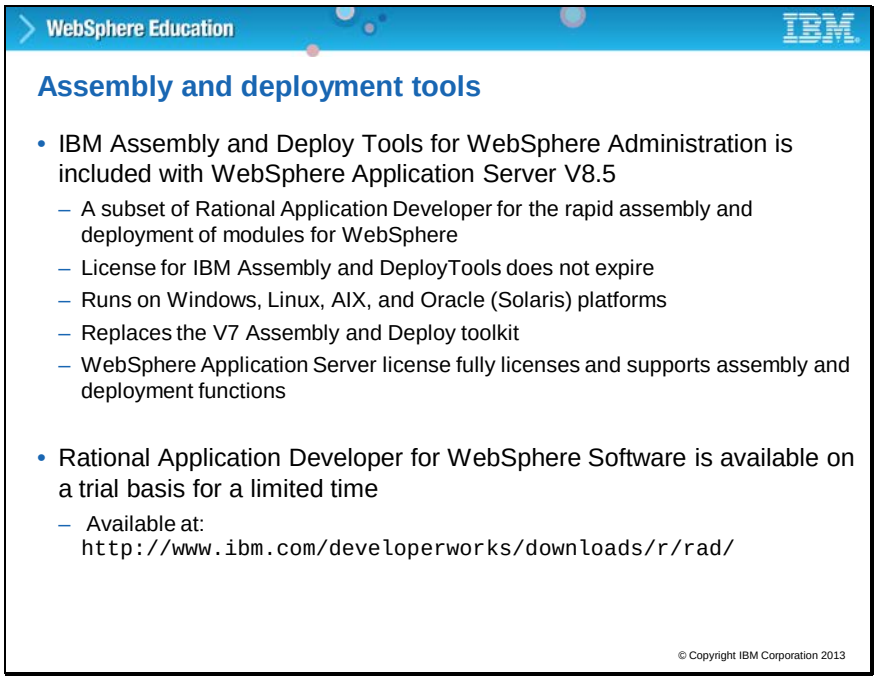
Slide 9



Topic: Application assembly and deployment tools

This topic describes the application assembly process and the tools that can be used.

Slide 10



Title: Assembly and deployment tools

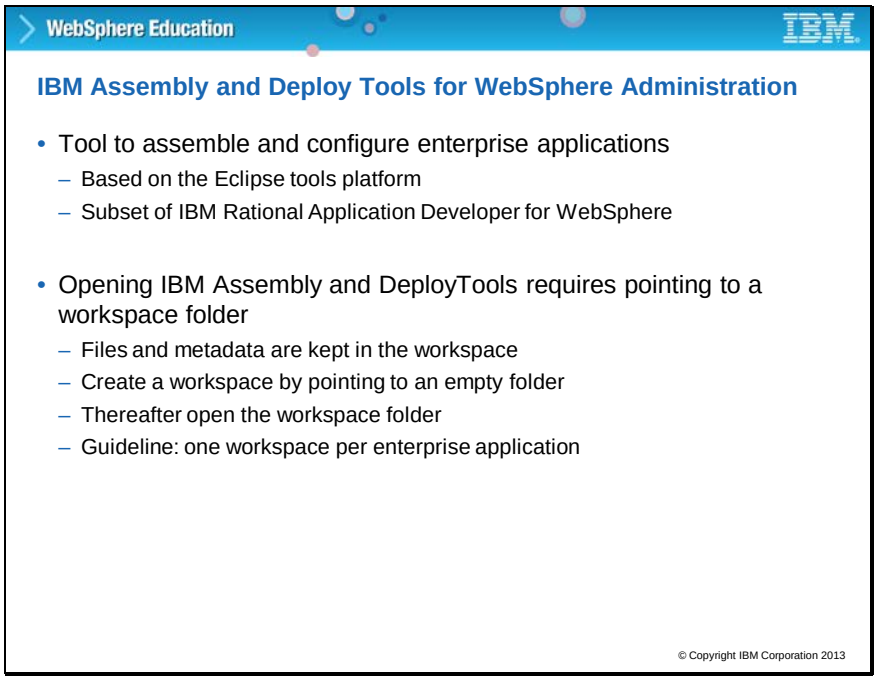
IBM Assembly and Deploy Tools for WebSphere Administration enable rapid assembly and deployment of applications to WebSphere Application Server environments. These tools replace the previously available IBM Rational Application Developer Assembly and Deploy function and are restricted to assembly and deployment usage only.

The IBM Assembly and Deploy Tools (IADT) are used to configure and package enterprise applications. It is based on the Eclipse platform and is a subset of the IBM Rational Application Developer for WebSphere. It includes source level debugging applications, a Jython editor, and source debugger, and you can use it to run wsadmin scripts inside the workspace. When starting the assembly and deployment tool, you can select either an empty workspace folder or an existing workspace folder.

Use the empty workspace folder if you are creating an EAR file from its individual components (WAR files or JAR files). After importing the components into the new workspace, in subsequent sessions, open the existing workspace folder.

Even if no modules are added to an empty workspace, the tool generates a “metadata” folder where it keeps its own internal data that represents the workspace. As modules are imported or added to the workspace, other folders are created to hold their contents.

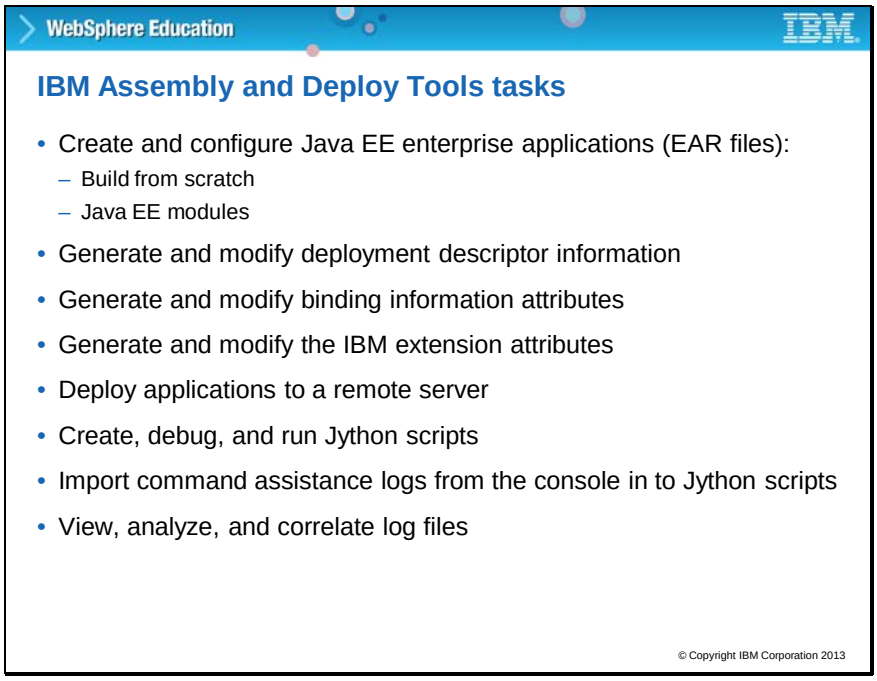
Slide 11



Title: IADT for WebSphere Administration

With IBM Assembly and Deploy Tools for WebSphere Administration, you get access to the complete set of Rational Application Developer documentation. Some documented features are only available with the full Rational Application Developer for WebSphere Software product.

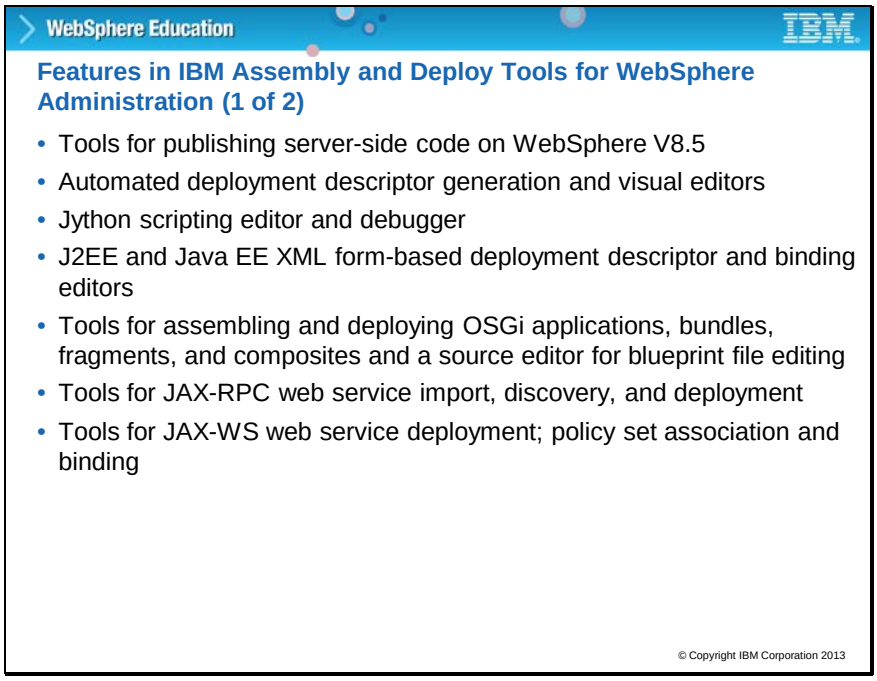
Slide 12



Title: IADT tasks

IADT provides all of the core functions that are listed on this slide for assembling and deploying a Java EE application.

Slide 13

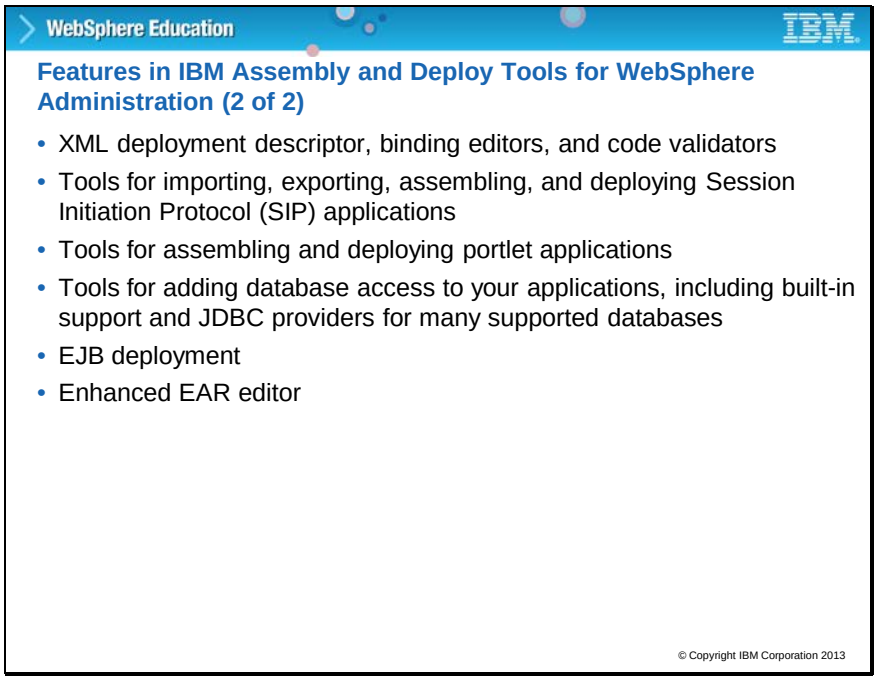


Title: Features in IADT for WebSphere Administration (1 of 2)

IADT provides tools for publishing server-side code on WebSphere V8.5 application servers, which include the following features:

- Automated deployment descriptor generation and visual editors.
- Jython scripting editor and debugger.
- J2EE and Java EE XML form-based deployment descriptor and binding editors.
- Tools for assembling and deploying OSGi applications, bundles, fragments, and composites.
- A source editor for editing a blueprint file.
- Tools for JAX-RPC web service import, discovery, and deployment.
- Tools for JAX-WS web service deployment; policy set association and binding.

Slide 14



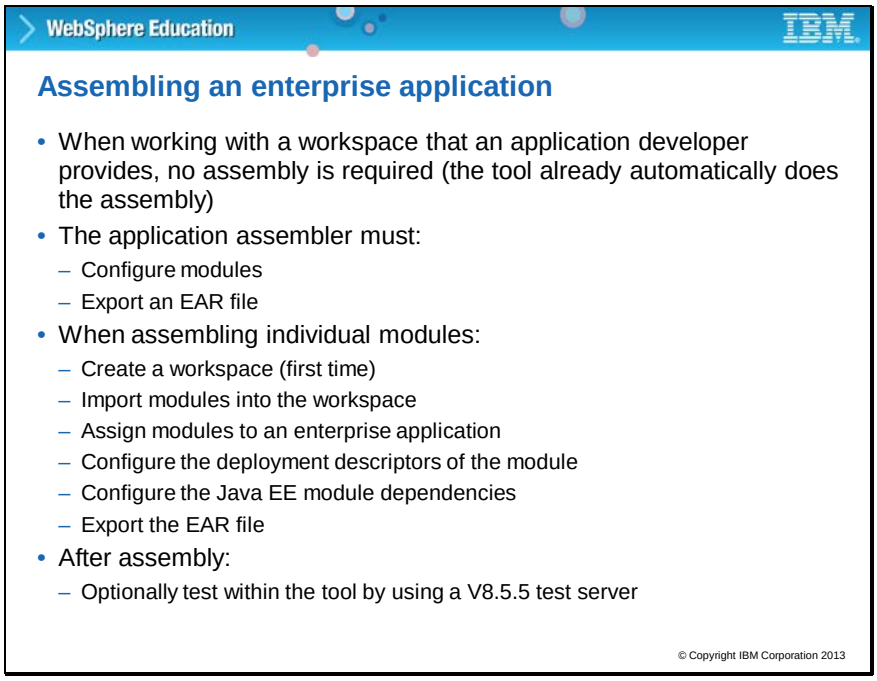
Title: Features in IADT for WebSphere Administration (2 of 2)

- XML deployment descriptor, binding editors, and code validators.
- Tools for importing, exporting, assembling, and deploying Session Initiation Protocol (SIP) applications.
- Tools for assembling and deploying portlet applications.
- Tools for adding database access to your applications, including built-in support and JDBC providers for many supported databases.
- EJB deployment.
- Enhanced EAR editor.

IADT supports the following WebSphere Application Server Version 8.x applications for assembly and deploy purposes:

- Java EE
- Basic OSGi
- Web Services
- XML
- Basic SIP
- Basic Portlet

Slide 15



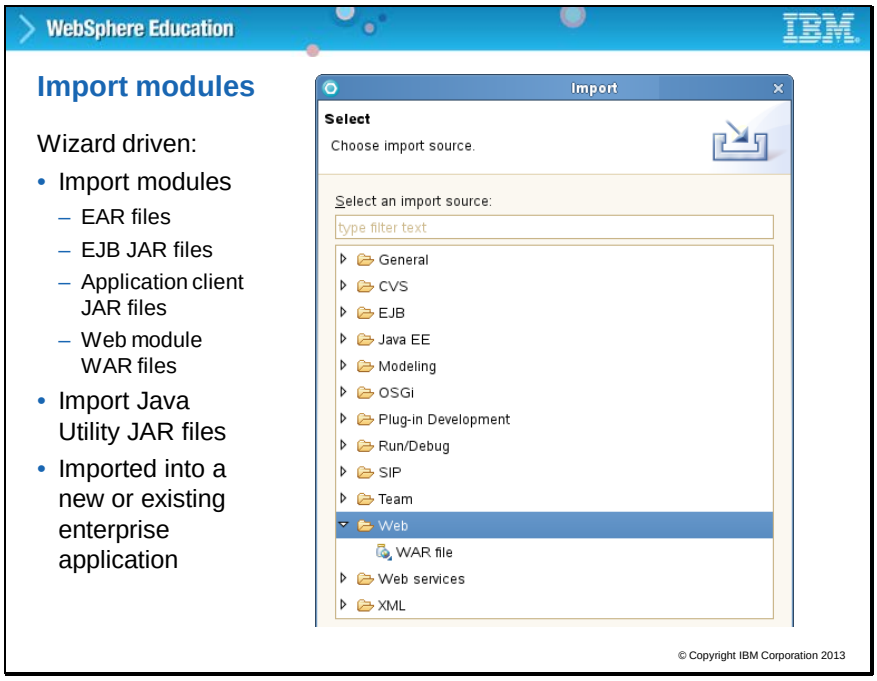
Title: Assembling an enterprise application

If your developers use IBM tools, you can receive an existing workspace folder for final configuration and deployment. In this case, the individual WAR and JAR files are not required since they are included in the workspace. You must point the tool to the root directory of the workspace.

If you receive the individual WAR and JAR files, you must point the assembly and deploy tool to an empty workspace and import the modules.

After assembly, you can test the application within the tool.

Slide 16

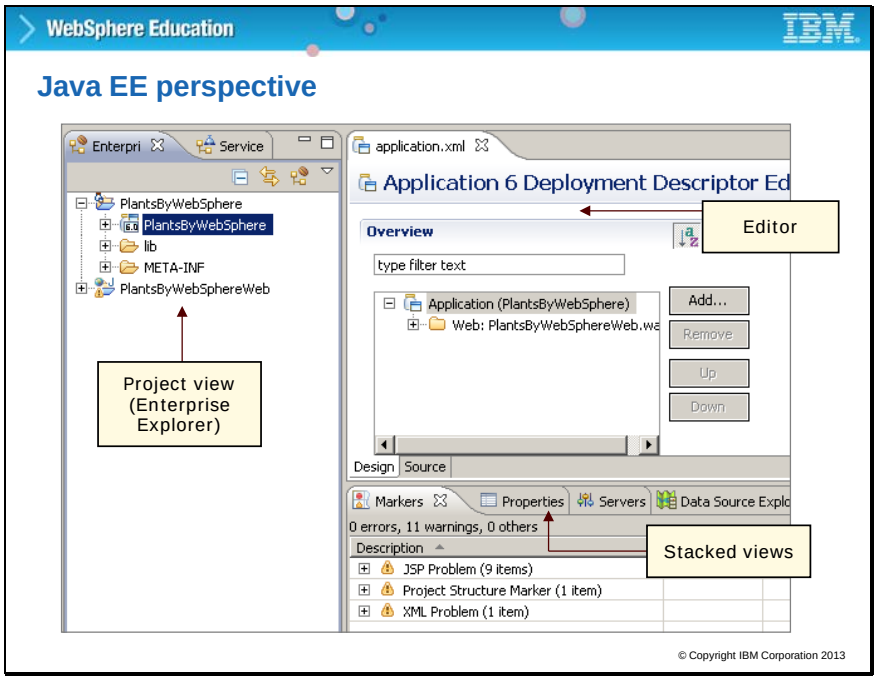


Title: Import modules

The import wizard is used to import all types of files into the workspace. The types that make up the modules of an enterprise application include an EAR file, EJB JAR files, WAR files, and others.

The Project Interchange file format can also be used to share entire or partial projects, which are special compressed files. When imported into an empty workspace folder, the Project Interchange file re-creates the tool metadata for the system it is being imported into. The Project Interchange format is an ideal format to share workspaces because workspace metadata is not stored within it.

Slide 17



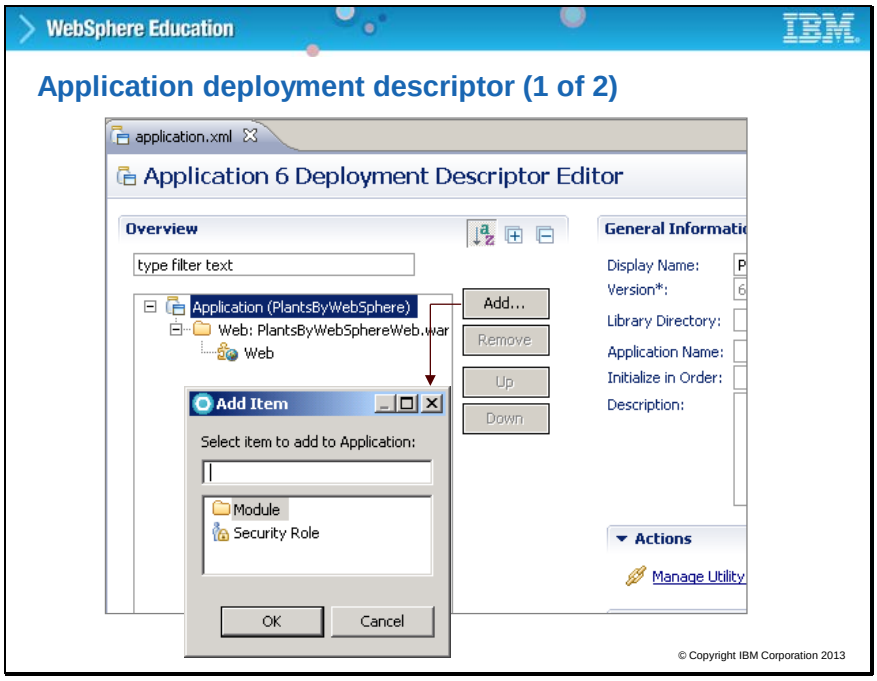
Title: Java EE perspective

This screen capture shows the primary perspective of IADT for packaging Java EE applications. As with the case of Rational Application Developer, the assembly and deployment tool is organized into perspectives, panes, views, and editors. Perspectives show a predetermined number of views to facilitate a role, like deployment.

Other useful perspectives are Server, Debug, Java, Web, and JPA. The Java EE perspective is used during the assembly, configuration, and deployment processes.

Panes hold views. Views can be stacked in panes. If stacked, views are accessible through tabs on the pane. Editors present data in a way that makes viewing and changing that data the easiest, based on the type of data. There are specialized editors for each of the deployment descriptor types, and for each of the file types that make up Java EE applications (Java, XML, and more).

Slide 18



Title: Application deployment descriptor (1 of 2)

The deployment descriptor for the application is shown in the source view of the customized editor. A recommendation is to use customized editors when working with deployment descriptors because they produce the correct XML without the risk of manual editing.

Use the tabs in the editor to access different sections of the file. The application.xml file defines all modules in the EAR file. It is important to note that it is not necessary to include an application deployment descriptor in a Java EE enterprise application.

If your enterprise application does not include the application deployment descriptor, you can optionally generate the application deployment descriptor stub from the menu of the enterprise application. In the assembly and deployment tool, right-click the enterprise project; then select Java EE and Generate Deployment Descriptor Stub from the menu.

Slide 19

WebSphere Education

Application deployment descriptor (2 of 2)

Simplest of deployment descriptors (`application.xml`)

The editor can be used to:

- Edit the display name and description of the application
- Add and remove modules
  - Web
  - EJB
  - Application client
  - Resource adapter (connector)
- Work with security roles of the application

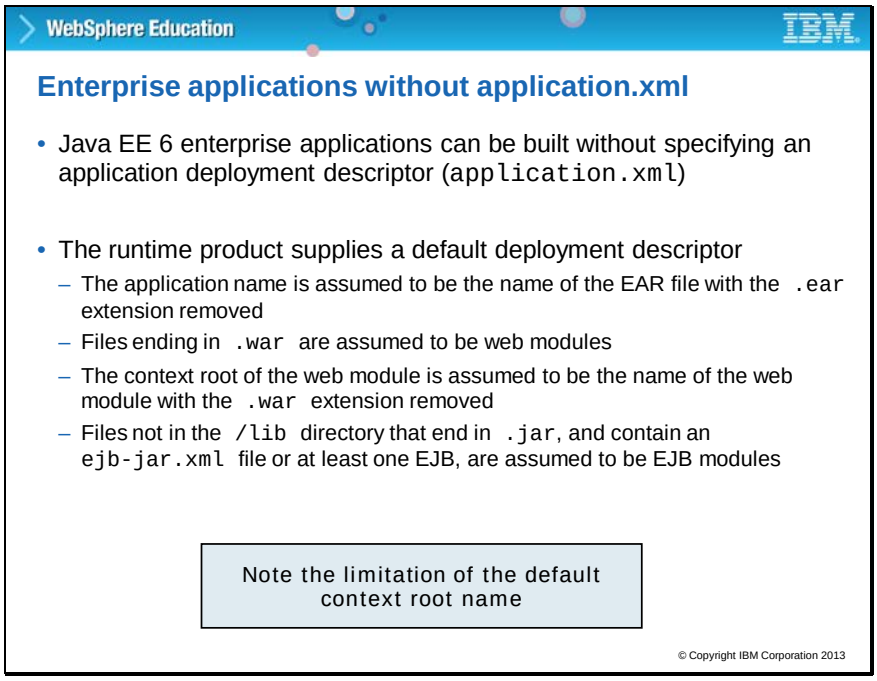
```
<?xml version="1.0" encoding="UTF-8"?><application id="Application_ID" version="6"
  ...
>
 <display-name>PlantsByWebSphere</display-name>
 <module id="Module_1320788080110">
            <web>
                        <web-uri>PlantsByWebSphereWeb.war</web-uri>
                        <context-root>PlantsByWebSphere</context-root>
            </web>
 </module>
   <security-role>
      <description>Samples Administrator</description>
      <role-name>SampAdmin</role-name>
   </security-role>
 </application>
```


Title: Application deployment descriptor (2 of 2)

As shown on the previous slide, the application deployment descriptor is the simplest of deployment descriptors. The editor can be used to edit the display name and descriptions, add, or remove modules, and work with security roles.

The application.xml file identifies each module of an application. As already mentioned, a Java EE 6 application is not required to provide an application.xml file in the EAR file. When an application.xml file does not exist, the product examines the Java archive (JAR) file contents to determine whether the JAR file is an enterprise bean (EJB) module or an application client module.

Slide 20



Title: Enterprise applications without application.xml

If the EAR file contains only the JAR and web archive (WAR) files, and no application.xml file, the product provides a default deployment descriptor. The default is based on the following criteria that are outlined in the Java EE specification.

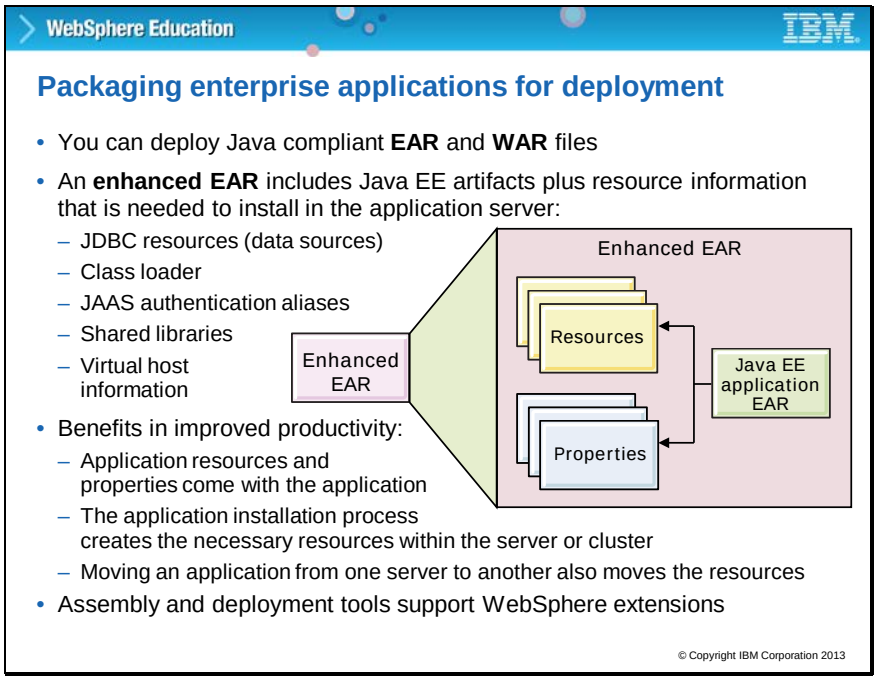
The application name is assumed to be the name of the EAR file, but with the .ear file extension removed. Files that end in .war are assumed to be web modules. The context root of the web module is the name of the file that is relative to the root of the application package, but with the .war file extension removed. This naming convention is used even if the context root is set in the web module properties in the assembly tool.

Files ending in .jar that are not in the /lib directory, and that contain either an ejb-jar.xml file or at least one class that defines a @Stateful or @Stateless annotation, are assumed to be EJB modules.

Other JAR files that are not in the /lib directory are not assumed to be EJB modules.

It is important to understand the limitation of the default context root name. For example, if the web module is named PlantsByWebSphereWeb.war, then the default context root is PlantsByWebSphereWeb.

Slide 21



Title: Packaging enterprise applications for deployment

An enhanced EAR file is created if you export an application by using the administrative console. To export applications, use the Export button on the Enterprise applications page. Using Export produces an enhanced enterprise archive (EAR) file that contains the application and the deployment configuration. The deployment configuration consists of the deployment.xml and other configuration files that control the application behavior on a deployment target.

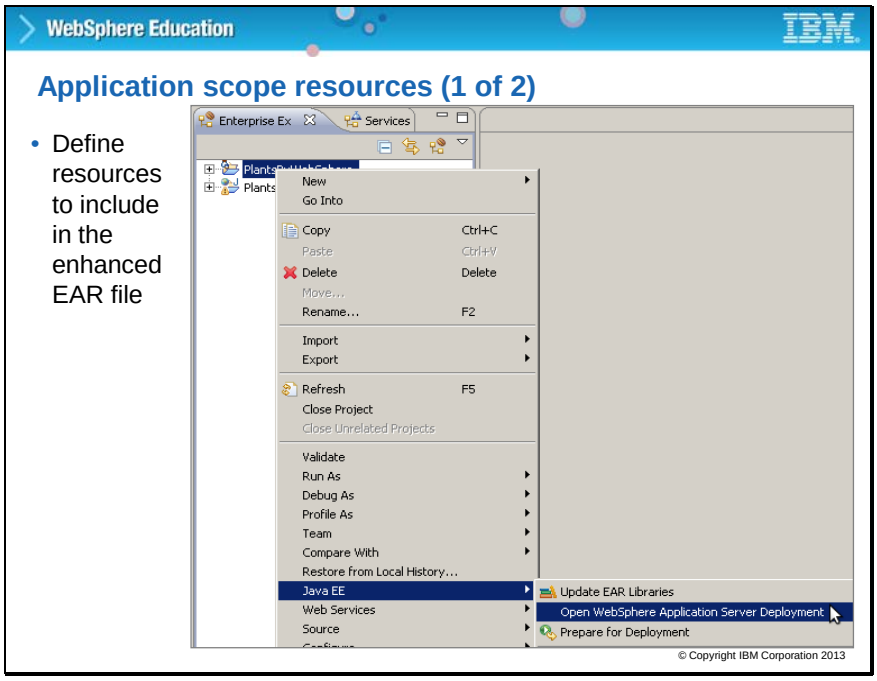
Exporting applications enables you to back up your applications and preserve binding information for the applications. You might export your applications before updating installed applications or migrating to a later version of the product.

Enhanced EAR files can include application-scoped resources, such as JDBC resources, class loaders, authentication aliases, and others. The application-scoped resources can be removed during application assembly, or ignored during application installation if they are not appropriate for the target server runtime environment. WebSphere extensions that are defined within an enhanced EAR file are ignored if this EAR file is installed on an application server other than WebSphere.

Developers and administrators can define resources and properties within an enterprise application and import or export enhanced EAR files. Some resources, such as JMS and JavaMail, still must be defined in the application server. Settings are defined in IBM tools and are

stored in deployment.xml and packaged with the EAR file. Resources are applied at the new application scope.

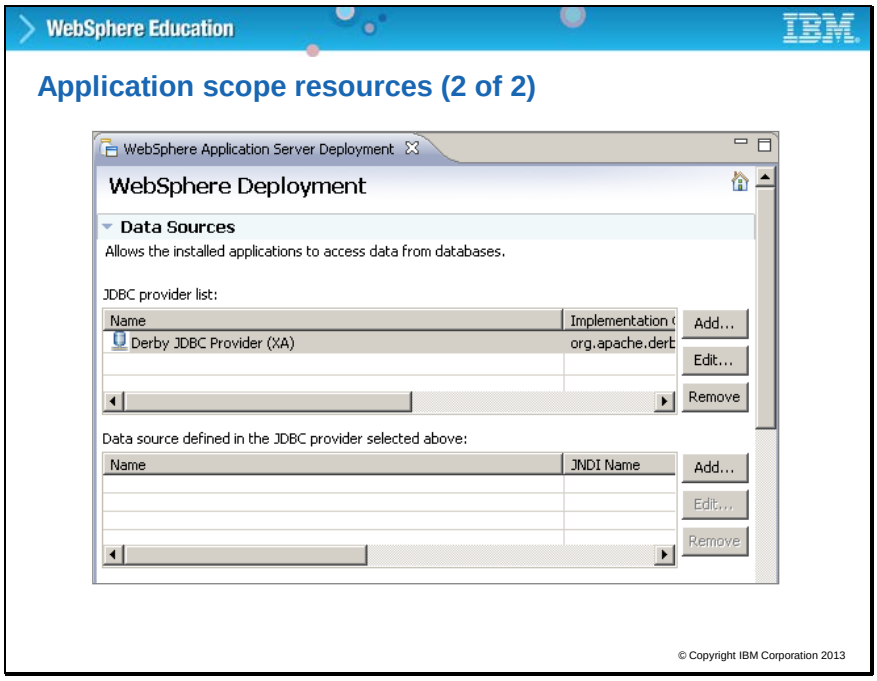
Slide 22



Title: Application scope resources (1 of 2)

This screen capture shows how to open the editor for resources that are going to be scoped at the application level. These types of resources are part of an Enhanced EAR. The next several slides provide more information about enhanced EARs.

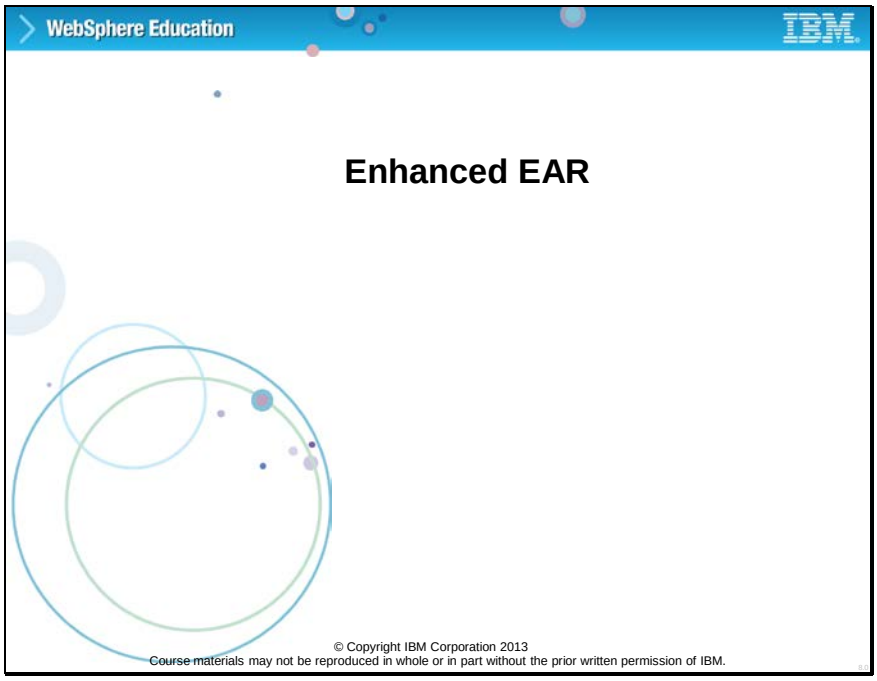
Slide 23



Title: Application scope resources (2 of 2)

In the WebSphere Application Server Deployment editor for the application, you can define certain resources that are included within the EAR file. Data sources, JAAS authentication aliases, class loader policy, shared libraries, and virtual host information can be defined on this page. Any resources on this page are defined at the new application scope. After deployment to and application server, any changes that are made in the administrative console are not saved to the EAR file. These changes are only saved to the server runtime configuration file.

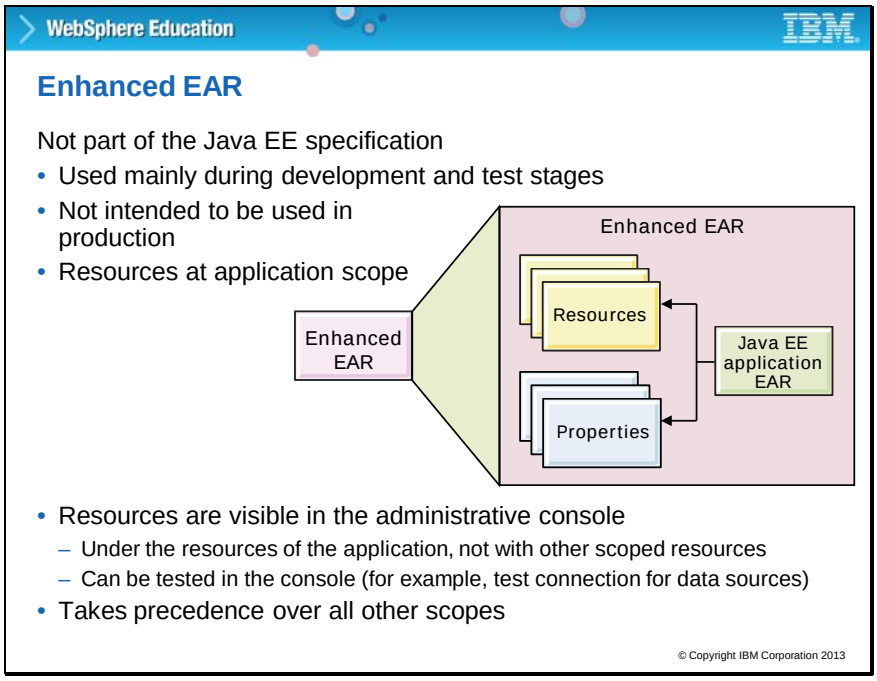
Slide 24



Topic: Enhanced EAR

This topic describes the enhanced EAR.

Slide 25

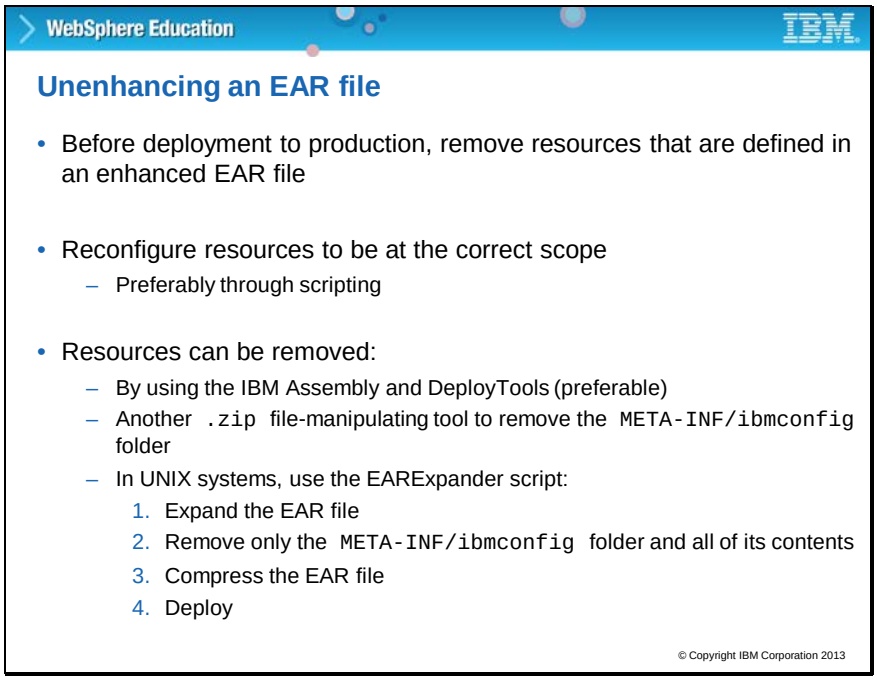


Title: Enhanced EAR

An enhanced EAR file is not part of the Java EE specification, but WebSphere uses it to hold configuration information mainly during development and test stages. In WebSphere Application Server, it is referred to as an embedded configuration.

Resources are defined at the application scope and are visible in the administrative console under the resources for the application, not with other scoped resources. The application scoped resources can be tested in the console (for example, test connection for data sources). Application scoped resources take precedence over all other scopes.

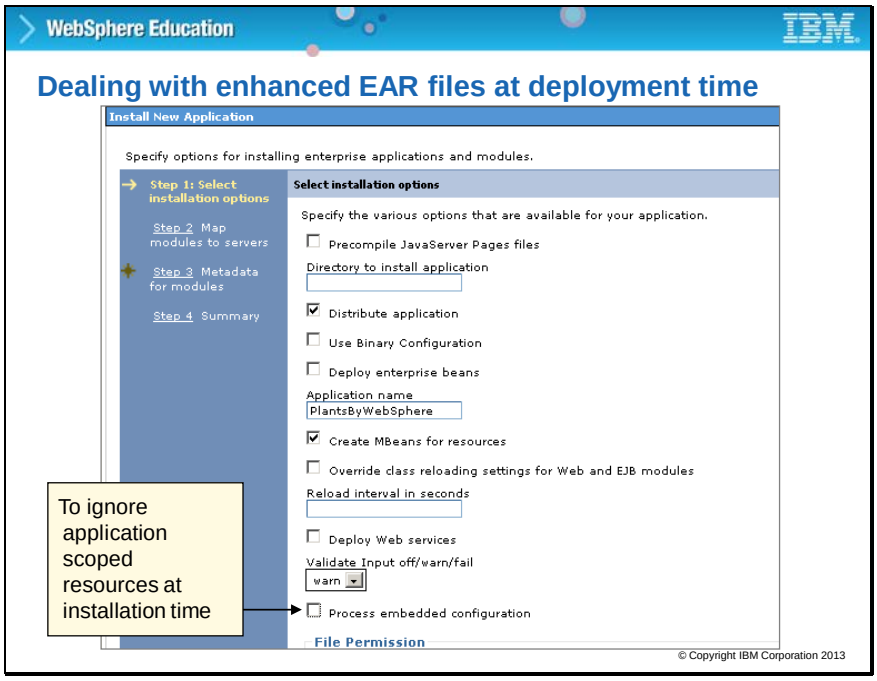
Slide 26



Title: Unenhancing an EAR file

Before deploying an application to production, resources in an enhanced EAR file must be removed and reconfigured at the proper scope, either in the administrative console or through wsadmin. When unenhancing an EAR file, make sure that you do not open the Deployment page of the deployment descriptor for the application. Doing so automatically creates an ibmconfig folder in the EAR file, even if nothing is ever put in there. The application server then behaves as if the EAR file is enhanced. To unenhance an EAR file, remove the META-INF/ibmconfig folder and its contents. Be careful to not remove the META-INF root folder. This process can most easily be done by using the assembly and deploy tool.

Slide 27

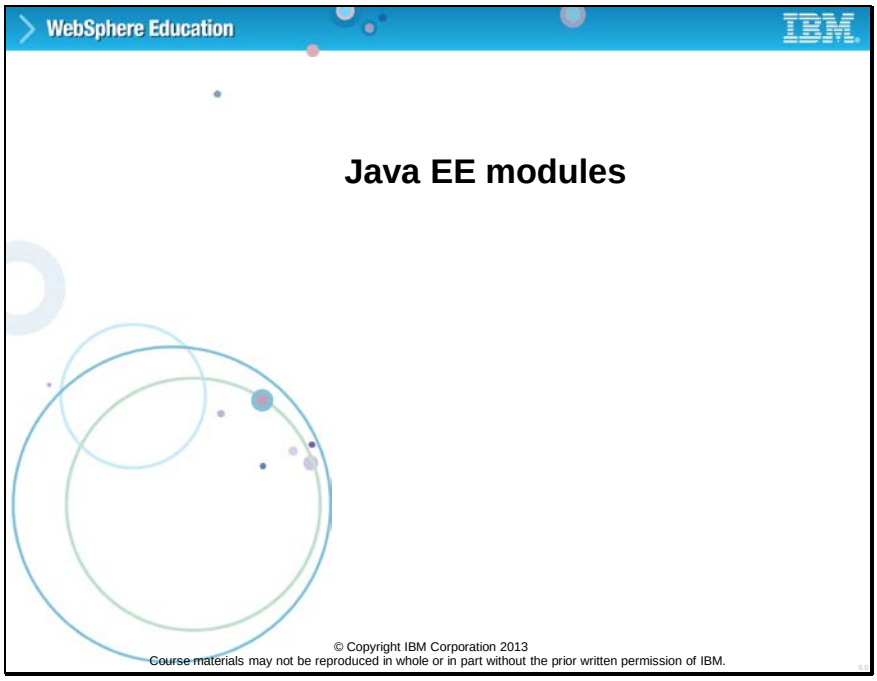


Title: Dealing with enhanced EAR files at deployment time

When an enhanced EAR file is installed on WebSphere Application Server, the “Process embedded configuration” box is checked. When this option is selected, the embedded configuration is loaded to the application scope from the .ear file. If the .ear file does not contain an embedded configuration, the “Process embedded configuration” box is cleared. You can choose to ignore the application scoped resources in the enhanced EAR file by clearing the “Process embedded configuration” option in the administrative console.

The check box named "Process embedded configuration" determines whether the application scoped resources are deployed to the server. If this box is checked, then the resources from the enhanced EAR get used. If it is not checked, these resources are ignored and not installed in the WebSphere Application Server.

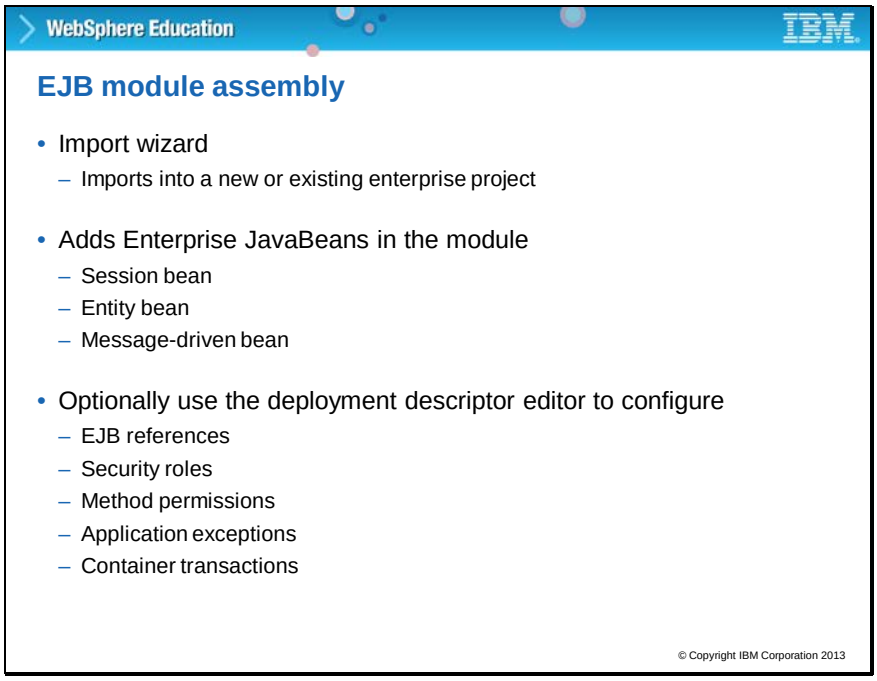
Slide 28



Topic: Java EE modules

This topic describes Java EE modules.

Slide 29



Title: EJB module assembly

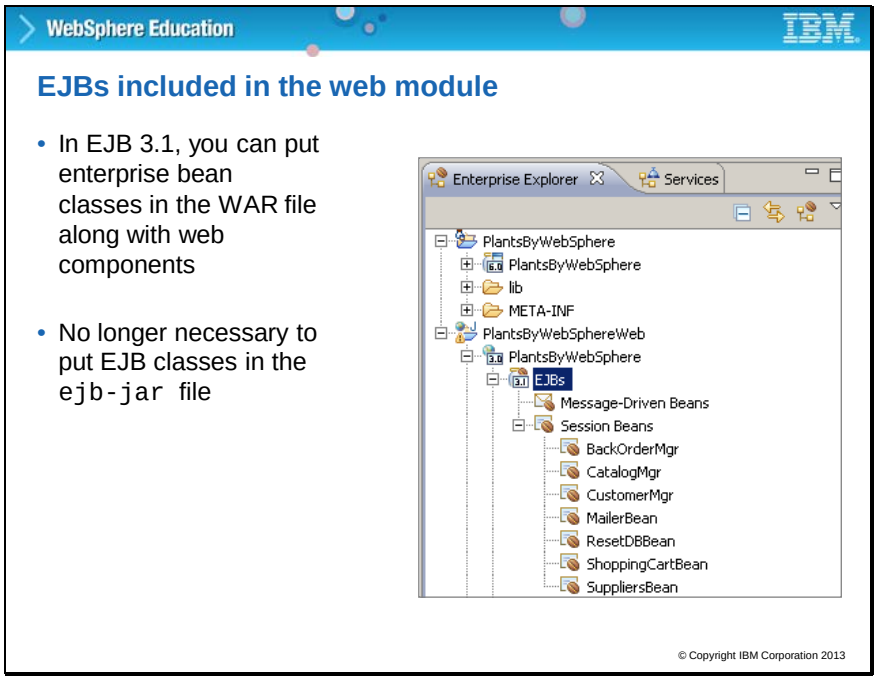
An enterprise bean is a managed Java component that can be combined with other resources to create Java EE applications.

You can assemble an EJB module to contain enterprise beans and related code artifacts. You can group web components, client code, and resource adapter code in separate modules. After the EJB module is assembled, you can install it as a stand-alone application or combine it with other modules into an enterprise application.

To begin the module assembly process, you can use the import wizard in the assembly and deploy tool to import a new or existing project and add modules to it. Optionally, you can modify the deployment descriptors by using the editor.

The EJB 3.0 specification provides annotation-based injection of EJBs into the program logic and no longer requires the use of an EJB deployment descriptor. For this reason, the application assembler is not concerned with making configuration changes to the EJB deployment descriptor (if one is created).

Slide 30

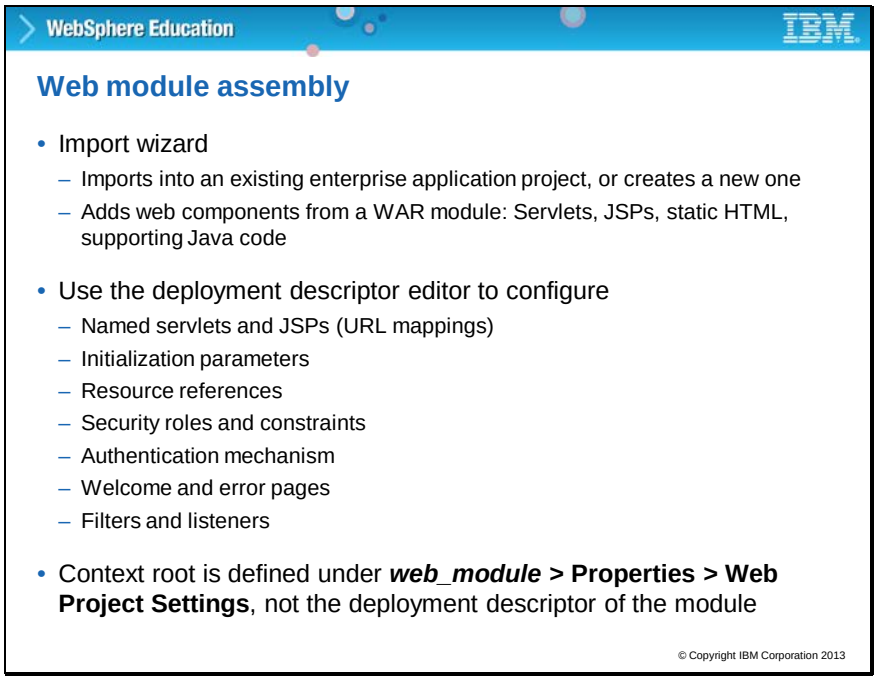


Title: EJBs included in the web module

In EJB 3.1, you can put enterprise bean classes in the WAR file along with web components. It is no longer necessary to put EJB classes in the ejb-jar file.

This screen capture from the IADT, shows an EJB JAR file that is packaged into a WAR module. In accordance with the Java EE 6 specification, EJBs can now be packaged into the WAR file.

Slide 31



Title: Web module assembly

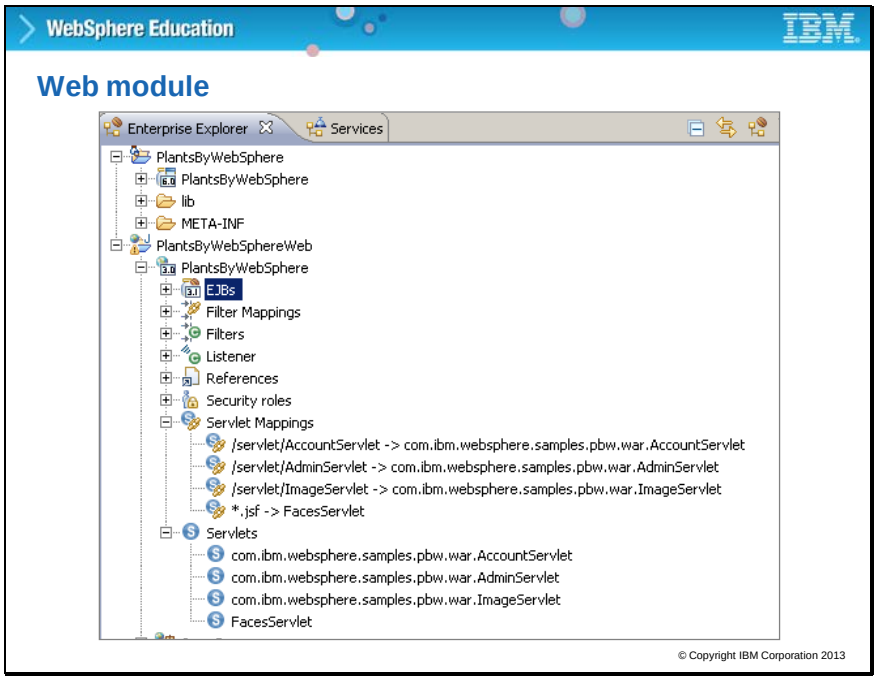
Web modules can be imported into new or existing web projects. The web projects can be part of new or existing enterprise applications. You can view or change the deployment descriptor by using the editor. The web module deployment descriptor includes the names of servlets, JSPs, URL mappings, initialization parameters, and resource references.

You can use the wizard to add web components from a WAR module such as: Servlets, JSPs, static HTML, and other supporting Java code.

When importing a WAR file that contains a web module into a web project, you are given a chance to change the existing context root for the module.

After importing the module, you cannot change the context root by using the deployment descriptor editor. To change the context root of a web project, you must open its properties and display the web page, where you can modify it.

Slide 32

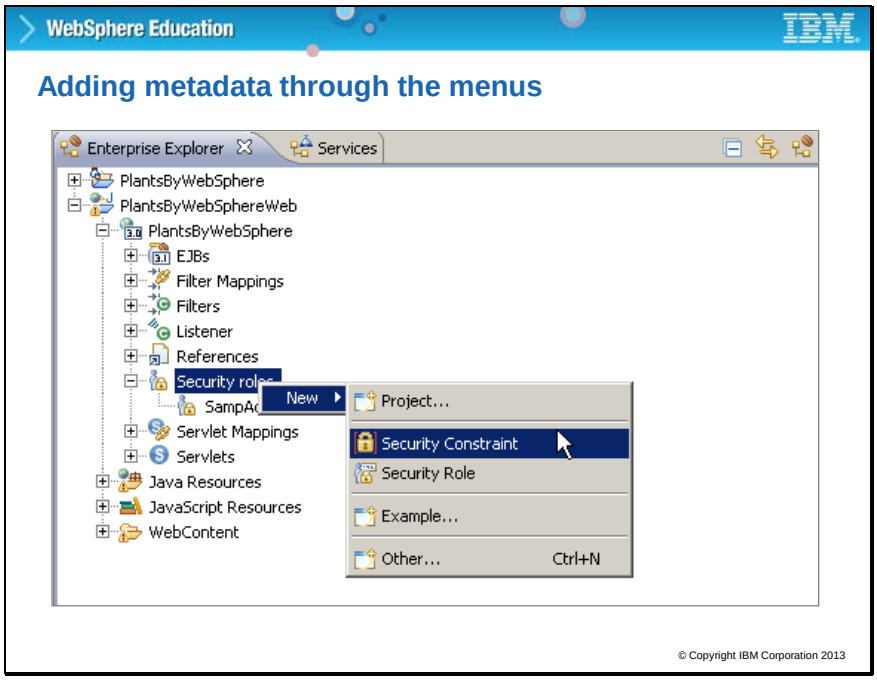


Title: Web module

In the Enterprise Explorer view, you can expand the web module to see all the components it contains. You can double-click some of these components, such as the servlet mappings, and that action opens the web deployment descriptor in the custom editor.

This screen shows some of the files that are contained in the web module. This module can be exported as a WAR file.

Slide 33



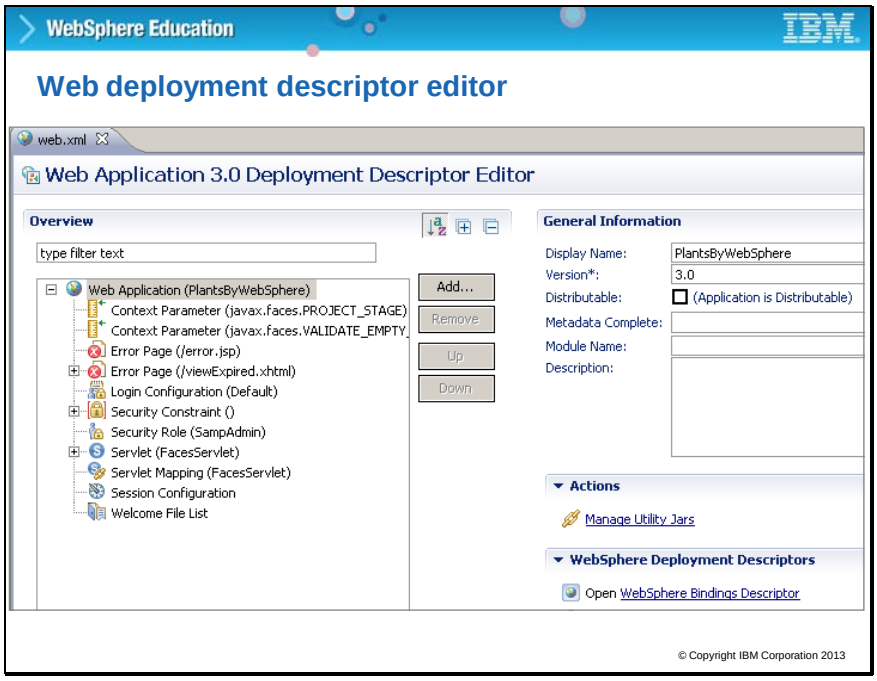
Title: Adding metadata through the menu

By clicking an item in the IADT, more information can be created or modified.

This screen capture shows the menu for adding metadata to the web project. In this example, right-click Security roles, then select New and Security Role to add a security role to the web application.

After entering the values for the security role, it is added to the web application deployment descriptor. Metadata refers to more program definition data that is attached to a program module either in external deployment descriptors or annotations in the program code itself. The metadata provides more context to the program.

Slide 34

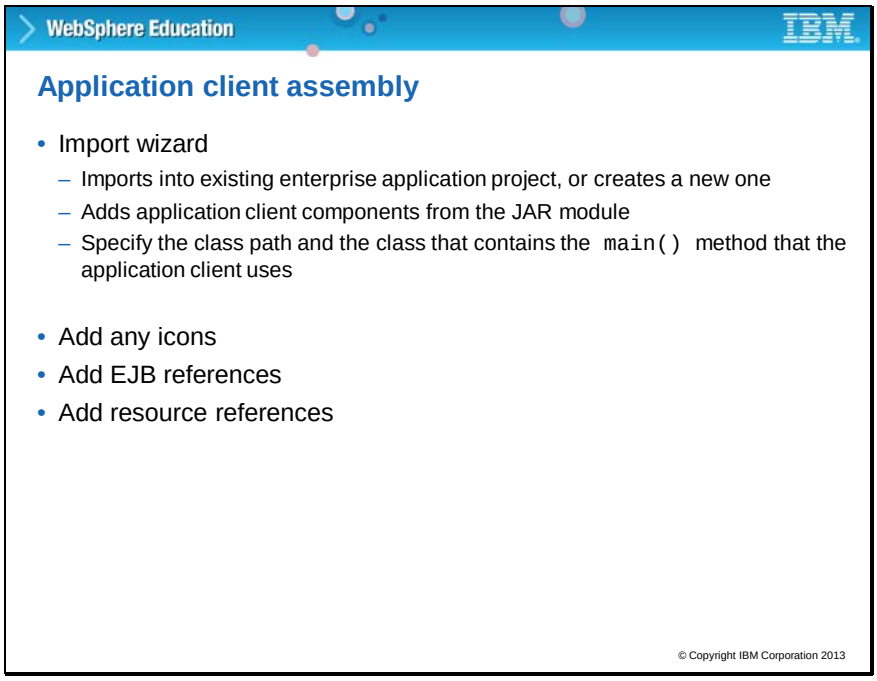


Title: Web deployment descriptor editor

This screen shows the web application deployment descriptor in the design view of the editor. This editor opens if you double-click the web.xml file or when you double-click a servlet mapping in the Enterprise Explorer view.

In general, you do not need to open the web deployment descriptor directly. You can use the menu for many items, as shown on the previous slide.

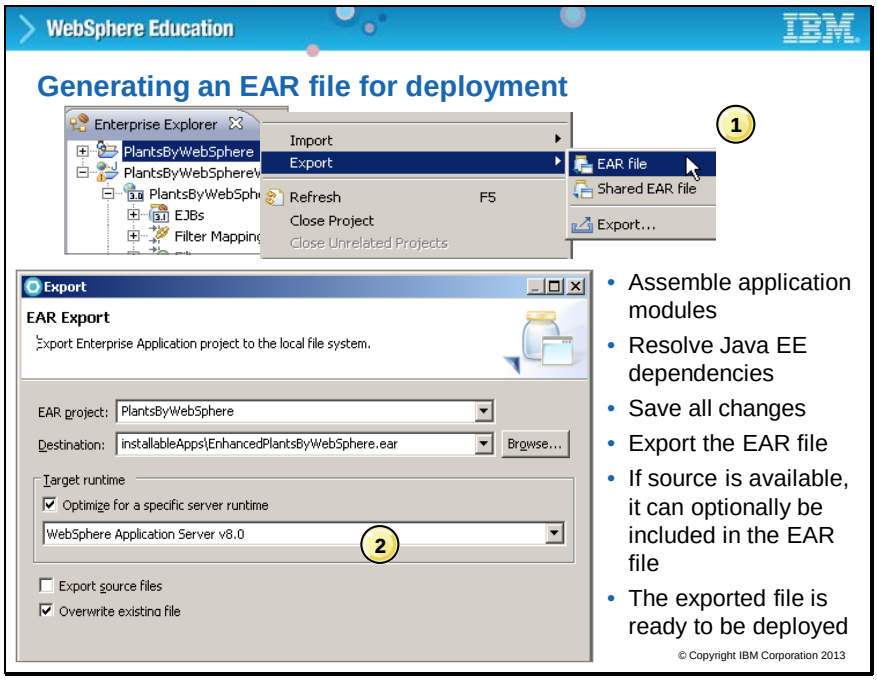
Slide 35



Title: Application client assembly

Importing a Java EE application client module is similar to importing the other types of modules. You choose the source file, the destination enterprise application, and client projects. The wizard does the rest. You can also include any icons, EJB references, and resource references, if applicable.

Slide 36

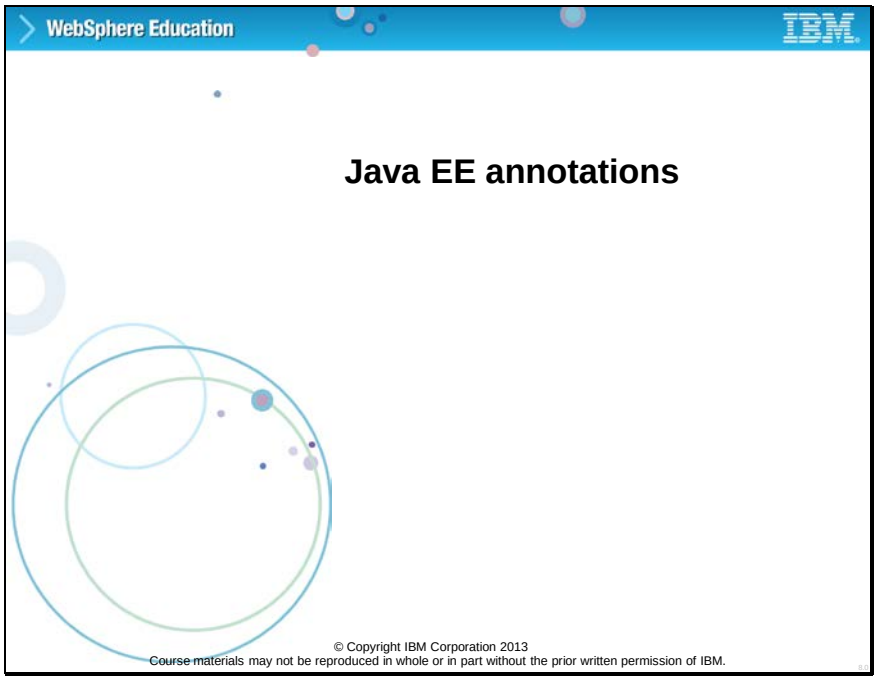


Title: Generating an EAR file for deployment

Enterprise applications are deployed in the form of an EAR file. Use the Export wizard to export an enterprise application project into an EAR file for deployment.

The wizard exports the contents of the EAR project to the specified EAR file. Additionally, for each project that corresponds to a module or utility JAR in the application, the project contents are exported into a nested module or JAR file in the EAR file. If any unsaved changes exist on any of the files in any of the referenced projects, you are prompted to save these files before export.

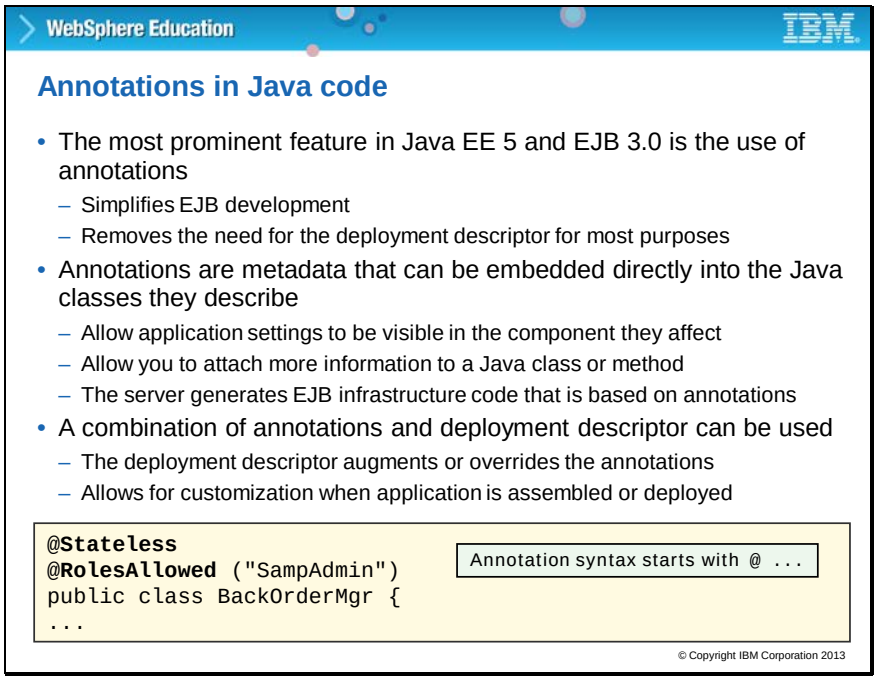
Slide 37



Topic: Java EE annotations

This topic describes Java EE annotations and how to use them.

Slide 38



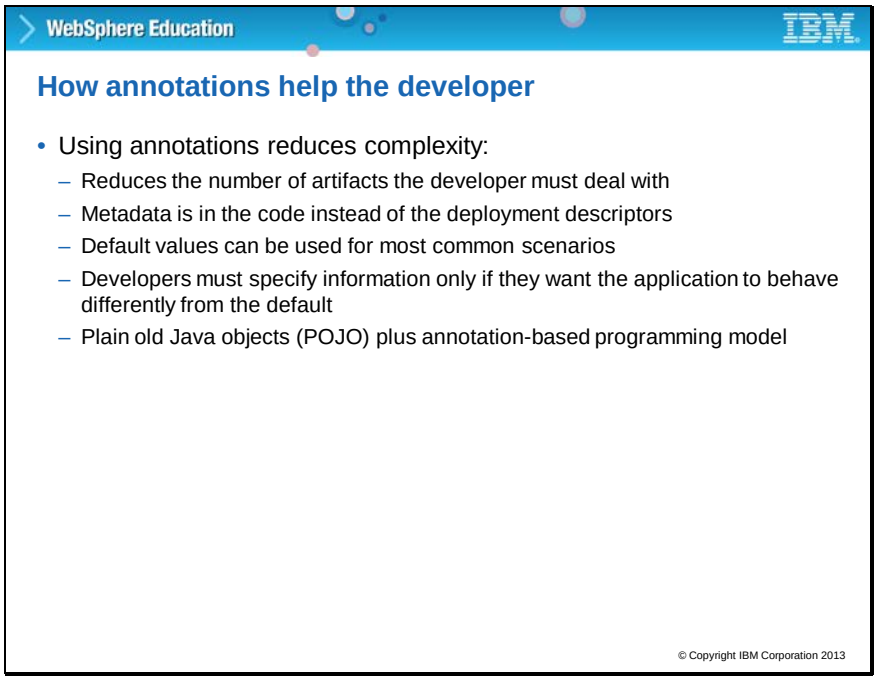
Title: Annotations in Java code

Annotations enable you to write metadata for EJBs inside your source code. You can use them instead of XML deployment descriptor files. Annotations can also be used together with descriptor files. The example code shows the syntax for the beginning of an annotation.

EJB 3.1 uses metadata annotations, which are part of Java SE 6. Java EE provides annotations for the following tasks:

- Defining and using web services
- Developing EJB software applications
- Mapping Java technology classes to XML
- Mapping Java technology classes to databases
- Mapping methods to operations
- Specifying external dependencies
- Specifying deployment information, including security attributes

Slide 39



Title: How annotations help the developer

Annotations help developers by reducing the complexity of their Java code, particularly for EJB applications. An EJB can now be an annotated "plain old Java object" (POJO) that does not need to extend a certain class. It must implement a remote interface only, which you define, or allow your development tool to create automatically. Deployment descriptors are no longer required because the EJB container can extract all that it needs from the annotations on an EJB.

Programmatic defaults eliminate the need to specify most commonly used requirements in the EJB container. You must specify an item if you do not want the default value. The dependency injection function and simpler lookup APIs, provide for easier access to an environment for the bean.

Slide 40

WebSphere Education

Example of declaring a session bean with annotations

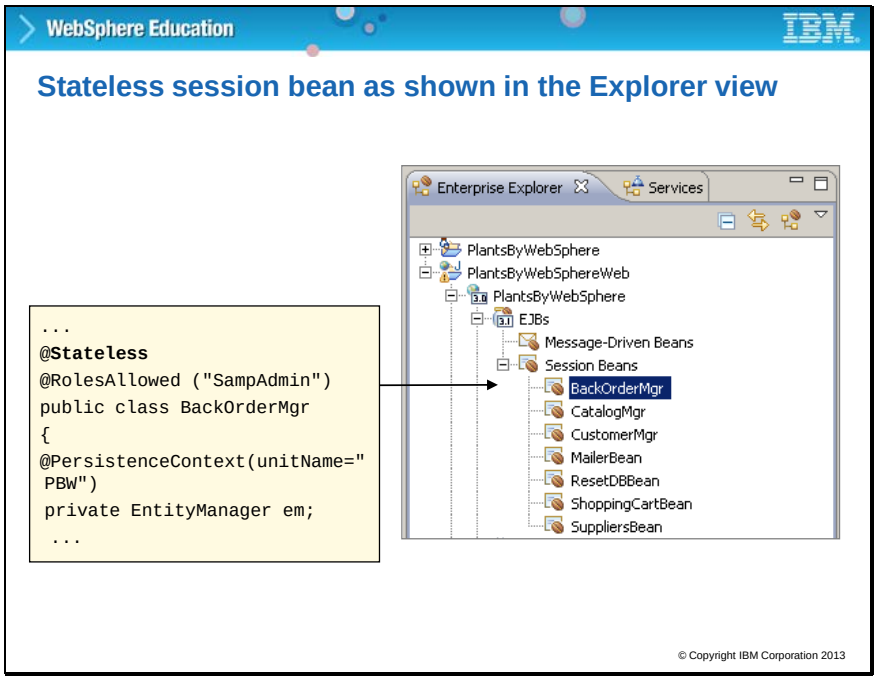
```
package com.ibm.websphere.samples.pbw.ejb;
import javax.ejb.Stateless;
 ...

/**
 * The BackOrderMgr provides a transactional and secured
 * facade to access back order information.  This bean no longer
 * requires an interface as there is one and only one implementation.
 */
@Stateless
@RolesAllowed ("SampAdmin")
public class BackOrderMgr
{
       @PersistenceContext(unitName="PBW")
       private EntityManager em;
 ...
```


Title: Example of declaring a session bean with annotations

This slide shows an example of how an EJB stateless session bean is declared by using annotations in the Java code.

In this example, the @Stateless metadata annotation declares the class BackOrderMgr as a stateless session bean. The annotation maps a security role that is called SampleAdmin to this session bean, and declares a persistence context for entity beans to use.

Slide 41



Title: Stateless session bean that is shown in Explorer view

The Enterprise Explorer view within the IADT allows you to see the hierarchy of the application, its modules, and the components of the modules.

This screen shows how the BackOrderMgr session bean, which is defined on the previous slide, displays in the list of session beans in the Enterprise Explorer view.

Slide 42

WebSphere Education

Using annotations to inject EJB references

- Dependency injection:
  - Hides creation and lookup of resources from application code
  - Resources are “injected” into the source code by the container that is based on the annotations specified
  - Programmers are no longer required to do JNDI lookups in their code
  - Clients that are managed classes can access EJBs by declaring a dependency:

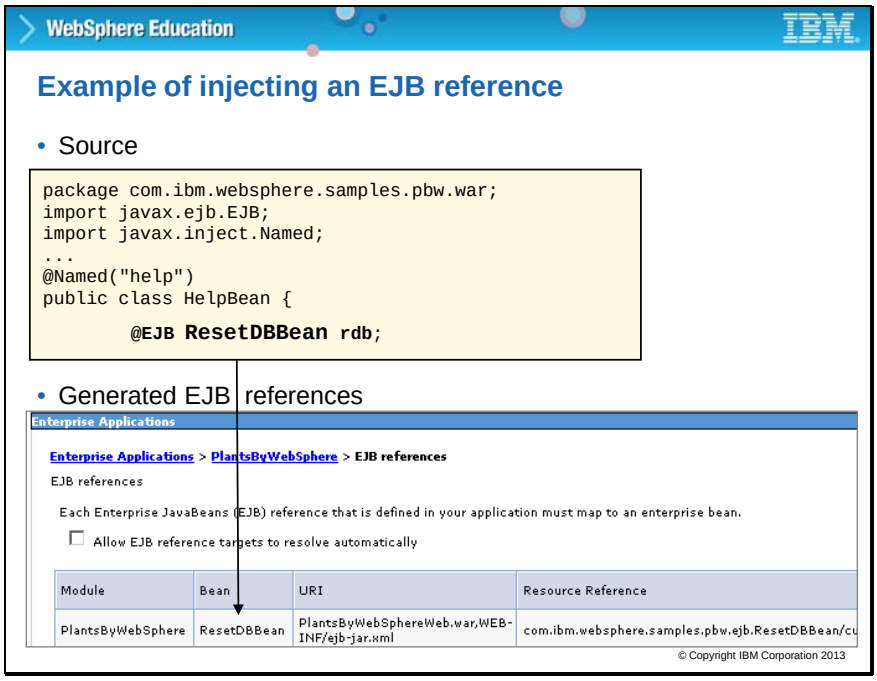
```
@EJB
  private CatalogMgr catalog;

  private ProductBean product;
  private LinkedList<ProductBean> products;
  private float shippingCost;
```


Title: Using annotations to inject EJB references

Dependency injection is an example of inversion of control. These terms both refer to the fact that the EJB container accesses objects for you based on annotations, by using JNDI to look up resources. These resources are injected into the source code. Dependency injection is used to hide resource creation and lookup from application code. Use the @EJB annotation to reference business interfaces of EJBs. After you have an EJB reference, you can immediately invoke the methods that the EJB interface defines. Using JNDI to locate and access EJBs is still available as an alternative to dependency injections.

Managed Java classes are classes that a container manages and include: EJBs, servlets, JSPs, servlet filters, event handlers, and Java classes in a client container.

Slide 43



Title: Example of injecting an EJB reference

The screen capture is from the administrative console. It shows a portion of the EJB references for the PlantsByWebSphere application. You can access this view by clicking Applications > Application Types > WebSphere enterprise applications > PlantsByWebSphere > EJB > EJB references

This example shows how a programmer injects an EJB reference into a Java program (HelpBean) by using annotation. The @EJB annotation in the example, has the same effect as adding an <ejb-local-ref> element to the ejb-jar.xml deployment descriptor and issuing a JNDI lookup on the bean. The EJB reference that the example shows is being injected into the HelpBean class in the PlantsByWebSphereWeb module of the PlantsByWebSphere enterprise application. You can view the generated EJB reference in the administrative console.

Slide 44

WebSphere Education

Application bindings

- All EJB references and resource references that are defined in the application must be bound to the artifacts defined in the application server
  - These artifacts include enterprise beans or resources
- You specify Java Naming and Directory Interface (JNDI) names for the referenced artifacts in an application
- For EJB 3.0 modules, binding definitions are stored in `ibm-ejb-jar-bnd.xml`
- For EJB 3.0 modules, you do not need to specify JNDI binding names for each of the business interfaces on your enterprise beans
  - If you do not explicitly assign bindings, the EJB container assigns default bindings

Title: Application bindings

An application assembler can define bindings when modifying deployment descriptors of an application. Bindings are specified in the WebSphere Bindings section of a deployment descriptor editor. Modifying the deployment descriptors might change the binding definitions in the ibm-ejb-bnd.xmi files that are created when developing an application. After defining the bindings, the assembler gives the application to a deployer. When installing the application onto a supported application server, the deployer does not modify, override, or generate default bindings unless changes are necessary for successful deployment.

Slide 45

WebSphere Education

WebSphere default bindings for bean interfaces

- WebSphere assigns default names that are based on patterns:

Description	Binding pattern
Short form local interfaces	`ejblocal:<package.qualified.interface>`
Short form remote interfaces	`<package.qualified.interface>`
Long form local interfaces	`ejblocal:<component-id>#<package.qualified.interface>`
Long form remote interfaces	`ejb/component-id># <package.qualified.interface>`

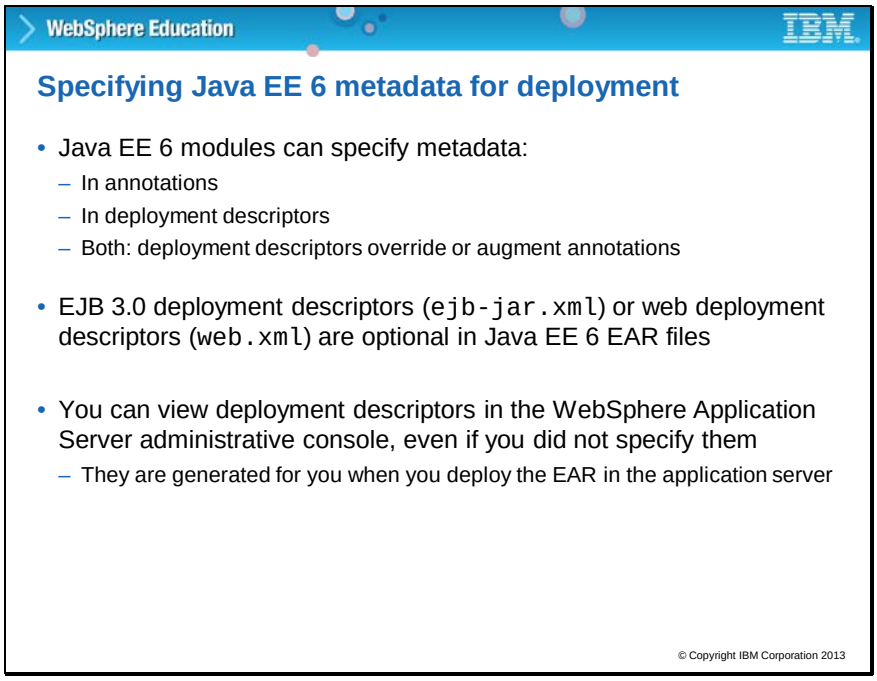
Title: WebSphere default bindings for bean interfaces

This table shows the WebSphere default EJB 3.x application bindings for bean interfaces. You must know these default bindings to write a client that uses JNDI lookup rather than dependency injection to access an EJB.

An application deployer or administrator can modify the bindings when installing the application onto a supported application server by using the administrative console. New binding definitions can be specified on the installation wizard pages.

Also, a deployer or administrator can select to generate default bindings during application installation. Selecting the “Generate default bindings” option during application installation instructs the product to define incomplete bindings in the application with default values. Existing bindings are not changed.

Slide 46

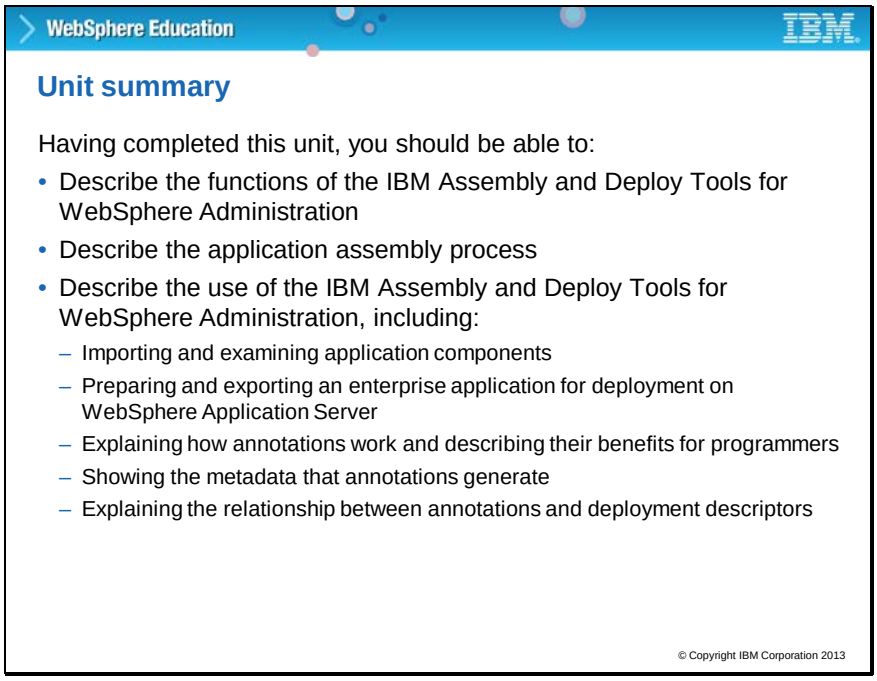


Title: Specifying Java EE 6 metadata for deployment

Support for bindings in the EJB container is expanded in Java EE 6. The EJB container assigns default JNDI bindings for EJB 3.x business interfaces that are based on the application name, the module name, and the component name. You do not have to explicitly define JNDI binding names for each of the interfaces or EJB homes within an EJB 3.x module or no-interface views within an EJB 3.1 module.

You can view deployment descriptors in the WebSphere Application Server administrative console, even if there are not any defined in the EAR file. They are generated when you deploy the EAR file to the application server.

Slide 47



Title: Unit summary

Having completed this unit, you should be able to:

- Describe the functions of the IBM Assembly and Deploy Tools for WebSphere Administration
- Describe the application assembly process
- Describe the use of the IBM Assembly and Deploy Tools for WebSphere Administration, including:
  - Using the IBM Assembly and Deploy Tools for WebSphere Administration to import and examine application components
  - Preparing and exporting an enterprise application for deployment on WebSphere Application Server
  - Explain how annotations work and describe their benefits for programmers
  - Show the metadata that annotations generate
  - Explain the relationship between annotations and deployment descriptors