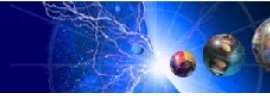


IBM WebSphere Application Server V4.0

WebSphere 4.0 Assembly and Deployment Tools



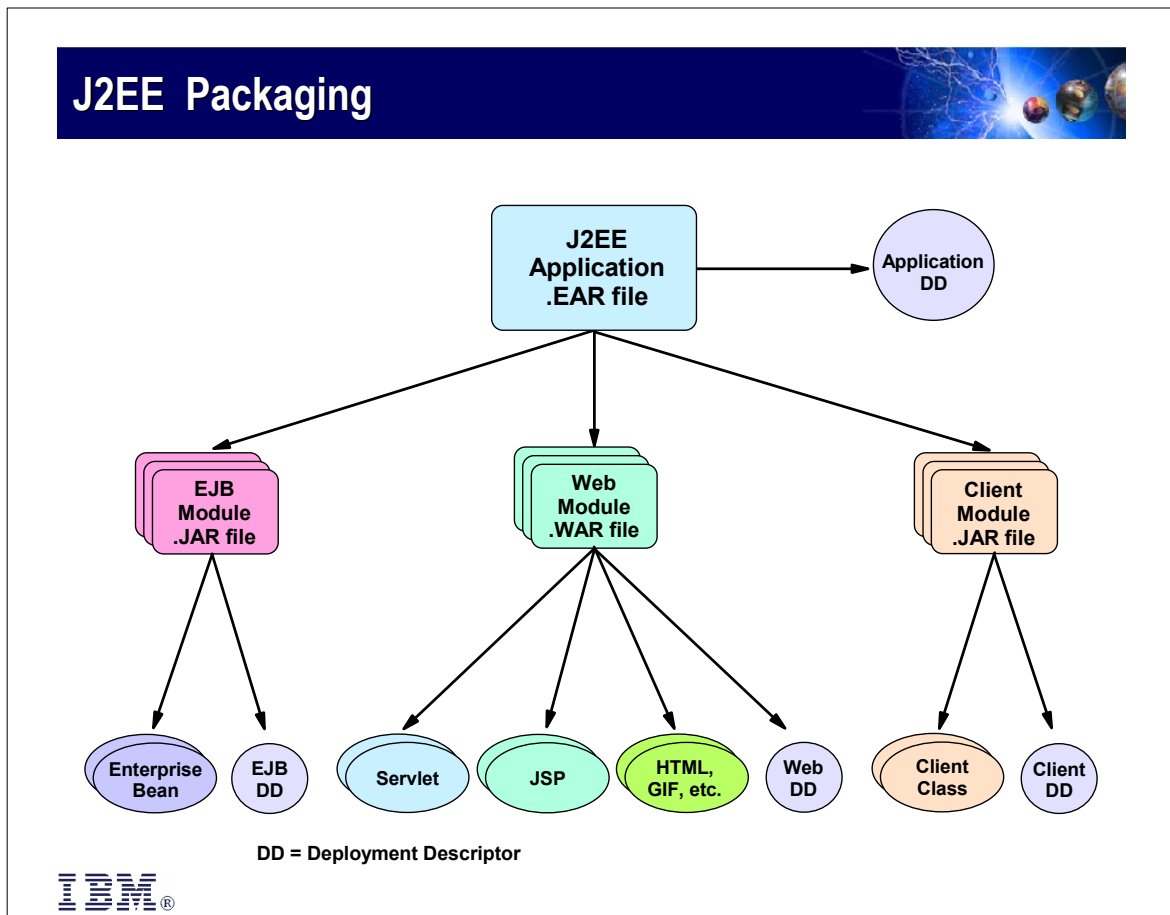
Topics



- Build J2EE application
 - ▶ Application Packaging
 - ▶ Application Assembly Tool (AAT)
 - ▶ ejbDeploy command line tool
 - ▶ EARExpander tool
- Installing J2EE Application
 - ▶ In WebSphere V4.0 AEs
 - Command line tool: SEAppInstall
 - Browser-based Admin Console
 - ▶ In WebSphere V4.0 AE
 - Standalone Admin Console
 - XMLConfig
 - WebSphere Control Program (WSCP)



J2EE Packaging



- ▶ A J2EE application is packaged in an Enterprise Archive, a file with a .EAR extension.
 - ▶ The application has a deployment descriptor, shown in the diagram as DD, allowing configuration to a specific container's environment when deployed.
 - ▶ The application can include one or more modules.
- ▶ J2EE components are grouped in modules, and each module has its own deployment descriptor.
 - ▶ EJB modules group related EJBs in a single module, and are packaged in Java Archive (JAR) files.
 - ▶ Note that there is only one deployment descriptor for all of the EJBs in the module. Previously, in WebSphere V3.5, each Enterprise bean had its own deployment descriptor.
 - ▶ Web modules group servlet class files, JSPs, HTML files, and images.
 - ▶ They are packaged in Web Application Archive (WAR) files.
 - ▶ Application client modules are packaged in Java Archive (JAR) files.

WebSphere Specific Packaging : IBM Bindings

- J2EE provides a mechanism to use local names for external EJB and Resource objects (for example, JDBC, JavaMail, JMS).
- The J2EE specification does not define how ejb-ref and resource-ref objects are tied into a J2EE runtime.
- WebSphere V4.0 defines *bindings* for this purpose.
 - ▶ Bindings are configured in AAT, often at the same time you define the resource-ref or ejb-ref
 - ▶ Bindings are stored in the JAR, WAR, or EAR file in a file named ibm-type-bnd.xmi



- ▶ Each J2EE vendor has its own implementation to resolve the EJB and resource references to the actual JNDI name. WebSphere V4.0 does it using IBM bindings.

ibm-web-jar-bnd.xmi

- ▶ Binds EJB reference to the EJB JNDI name
- ▶ Binds Resource References to JNDI name

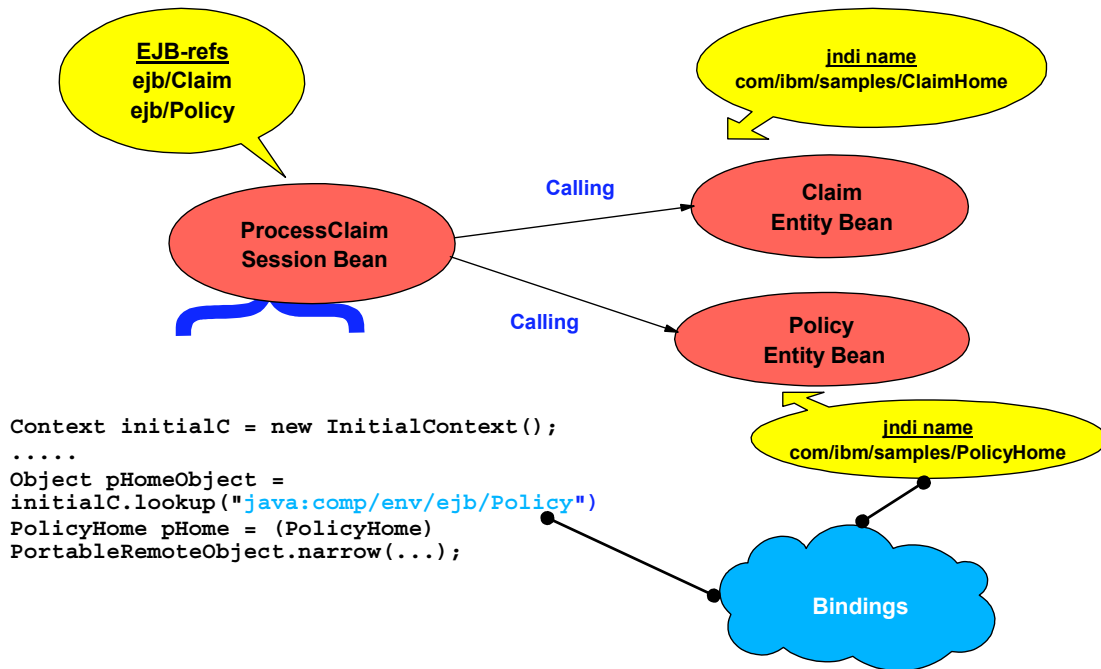
ibm-ejb-jar-bnd.xmi

- ▶ Binds EJB to JNDI name
- ▶ Binds EJB reference to the its JNDI name
- ▶ Binds Resource References to its JNDI name

ibm-application-bnd.xmi

- ▶ Binds security roles to users, groups, and special-subjects

WebSphere 4.0: Binding Example



- ▶ The Application Component provider can refer to the homes of the enterprise beans using "logical" names called *EJB references*. The EJB references are special entries in the naming environment of the application component. The Deployer binds the EJB References to the enterprise bean homes in the target operational environment.
- ▶ In the example, the Application component provider assigned the environment entry `ejb/Policy` as the EJB Reference name to refer to the home of the Policy Enterprise bean. At deployment time, this reference will be resolved using the IBM bindings.

WebSphere Specific Packaging: IBM Extensions

- Additional WebSphere options not included in the J2EE specifications:
 - ▶ Transaction isolation attributes
 - ▶ Web application reloading
 - ▶ File Serving and Servlet Invoker (by classname)

- Defined as IBM extensions
 - ▶ Configured in AAT
 - Extensions tab of the component property notebook
 - ▶ Stored in the JAR, WAR, or EAR file in a file called `ibm-type-ext.xml`



▶ File Serving and Servlet Invoker (by classname) is not recommended in production

ibm-ejb-jar-ext.xml specifies IBM extension attributes, such as:

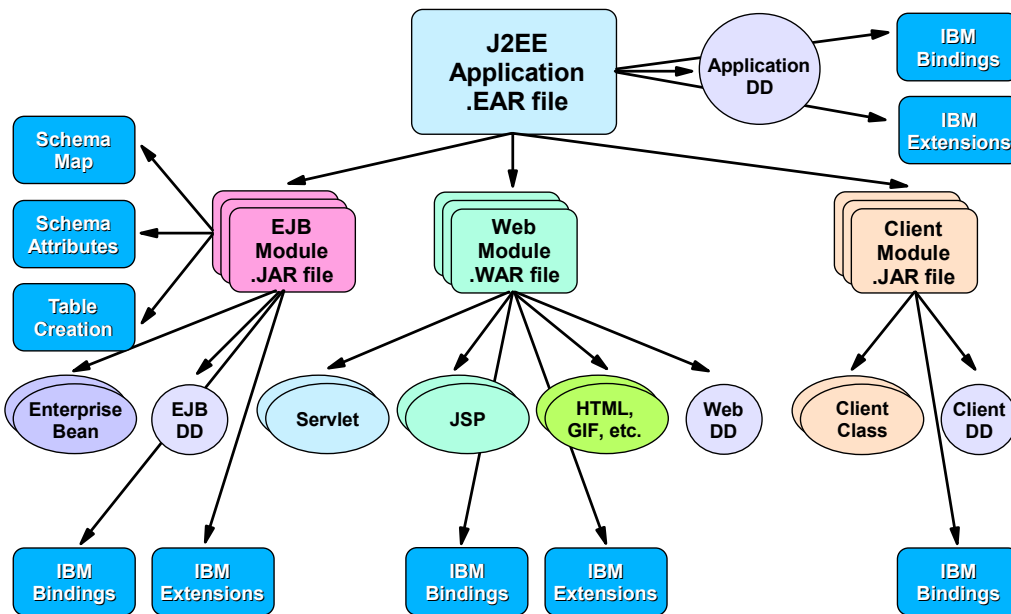
- ▶ Delegation (Run-As) mode for methods
- ▶ Transaction isolation level for methods
- ▶ Bean cache
- ▶ Local transaction

ibm-web-jar-ext.xml specifies IBM extension attributes, such as:

- ▶ Reload Interval
- ▶ Default Error Page
- ▶ File Serving Enabled
- ▶ Directory Browsing Enabled

ibm-application-ext.xml specifies type of modules and any extensions for WebSphere

WebSphere V4.0: Application Packaging



DD = Deployment Descriptor



- ▶ This page shows the J2EE Application EAR file enhanced with the IBM Bindings and Extensions.
 - ▶ These adapt the generic J2EE application to the IBM WebSphere V4.0 Application Server environment.
 - ▶ Most J2EE server vendors have their own proprietary extensions to the specification.
- ▶ The schema map, attributes, and table creation code provide the setup for entity beans to store their values in a database.
 - ▶ These are for entity beans that use Container-Managed Persistence, or CMP.
- ▶ All of these objects in the Application EAR file are packaged by the Application Assembly Tool.

Application Assembly Tool (AAT)

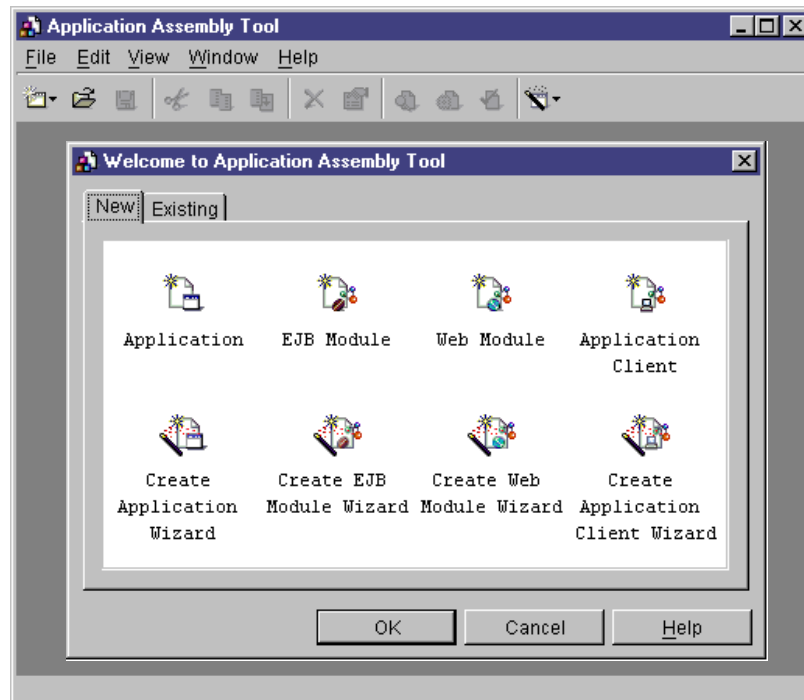


- Things you can do with AAT
 - ▶ Create/Edit J2EE Applications (EARs) from J2EE modules
 - ▶ Create/Edit J2EE modules
 - Web modules (.war files) servlets, JSPs
 - EJB modules (EJB .jar files)
 - Application Clients (.jar files)
 - ▶ Modify the deployment descriptor information
 - ▶ Modify the binding information attributes
 - ▶ Modify the IBM Extension attributes



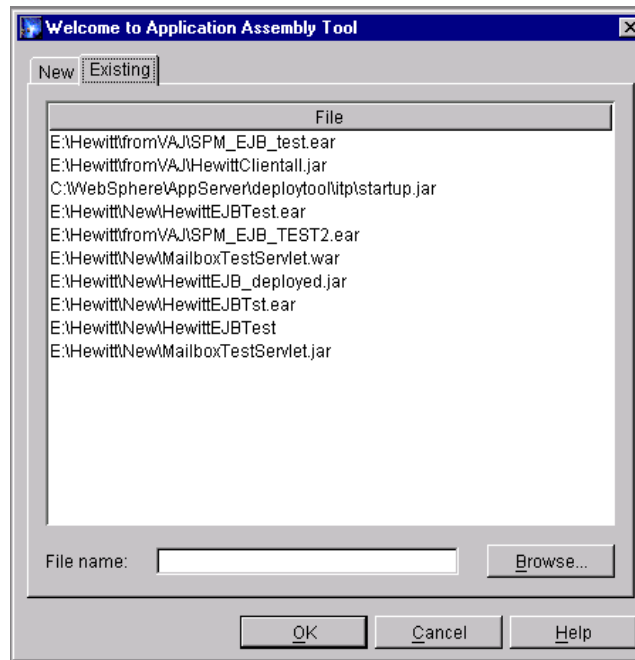
- ▶ Application Assembly Tool (AAT) is a GUI-based tool for assembling applications from application components and modifying their J2EE deployment descriptors.
- ▶ You can use AAT to create the J2EE ear from individual J2EE modules.
- ▶ There are three kinds of J2EE modules: Web modules, EJB modules and Java Client Application modules.
 - ▶ A Web module is used to assemble servlets, JSP files, Web pages and other static content into a single deployable unit.
 - ▶ An EJB module is used to assemble Enterprise JavaBeans into a single deployable unit.
 - ▶ An application client module is used to assemble the files that make up the application client into a single unit.
- ▶ AAT can be used to modify the deployment descriptor information, binding information, and IBM extensions. Each module as well as the enterprise application has deployment descriptors, bindings, and extensions stored in XML files.

Open the Application Assembly Tool



- ▶ Application Assembly Tool (AAT) can be started by running the `assembly.bat` file in Windows operating systems, or by running `assembly.sh` on Unix systems.
- ▶ The script is located in the directory where the WebSphere Application Server is installed under the "bin" subdirectory.
- ▶ Once AAT is started, you can quickly begin creating a new module using a wizard or opening an existing module.
- ▶ WebSphere Administrative Server does not need to be running to launch or use AAT.
- ▶ If the tool cannot be started, verify that the `assembly.bat` file is in your path or specify the fully qualified path to the script.
- ▶ To exit the tool, choose File->Exit.

AAT - Configuration File

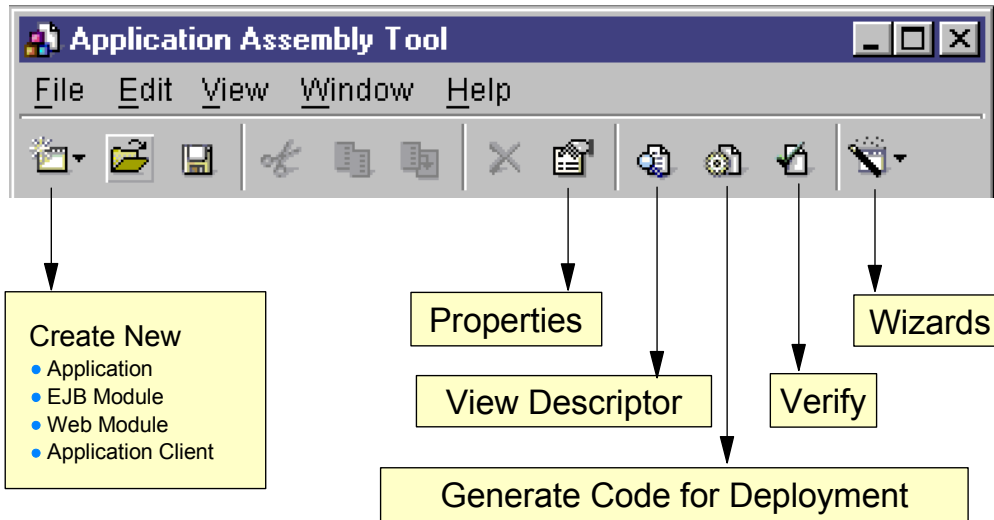


x:\winnt\profiles\\.AATConfig



- ▶ AAT fails to start if .AATConfig file corrupted. Delete this file to restart AAT.
- ▶ On UNIX, file is generated in home directory for the user.

AAT - Toolbar



AAT: Converting EJB JAR files from 1.0 to 1.1



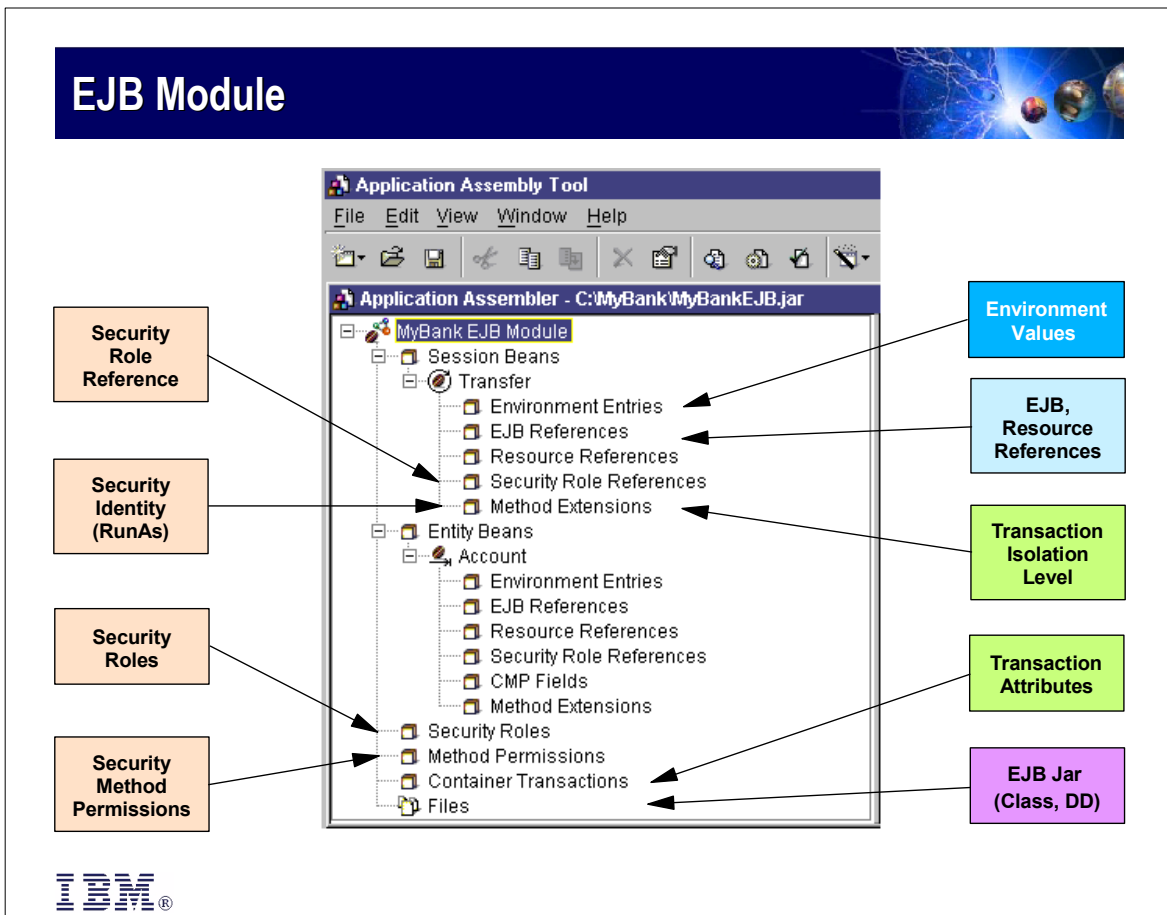
- Open the EJB 1.0 .jar file
- Save it using the Save As option
- The resulting .jar file will contain the EJB 1.1 DD and WebSphere extensions:
 - ▶ ejb-jar.xml
 - ▶ ibm-ejb-jar-ext.xmi
 - ▶ ibm-ejb-jar-bnd.xmi
 - ▶ Schema files, if necessary
 - Schema.rdbxmi
 - Map.mapxmi
 - Table.ddl



- ▶ AAT is somewhat friendly to Java technology preceding IBM WebSphere Application Server Version 4.0. For example, if you have an EJB 1.0 JAR file and a standalone Java client, you can turn it into a J2EE application by importing the JAR files into the assembly tool.
- ▶ The tool can help convert your EJB 1.0 (serialized) deployment descriptor to an EJB 1.1 XML deployment descriptor.
- ▶ You will probably need to make some code changes to make your EJB 1.0 application comply with J2EE.
- ▶ Read the 1.0 EJB into AAT. Save the file using File-->Save As. Look at the resulting .jar file. It will now contain the following files:

META-INF/MANIFEST.MF
META-INF/ejb-jar.xml
META-INF/ibm-ejb-jar-ext.xmi
META-INF/ibm-ejb-jar-bnd.xmi

EJB Module



- ▶ **Environment Entries** define variables used to customize the business logic of an application at runtime. This eliminates the need to access or change the application source code.
- ▶ An **EJB Reference** is a logical name used to locate the home interface of an enterprise bean used by the application. At deployment, the EJB reference is bound to the enterprise bean home in the target operational environment. If the session bean, Transfer, needs to access some other EJB, all the assembly properties for that EJB reference such as Name, Home Interface, Remote Interface and JNDI binding has to be provided. A resource reference declares a logical name used to locate a connection factory object.
- ▶ **Security Role References** specify the name of a security role reference used in the Application code.
- ▶ **Method Extensions**, stored in the `ibm-ejb-jar-ext.xmi` file, is the next item under the Transfer bean. The Transaction Isolation Level and Security Identity Run-As-Mode are specified here as extensions beyond the standard J2EE descriptors.
- ▶ Under **Assembly Properties** are three sub-folders: Security Roles, Method Permissions and Container Transactions. Security Role specifies the name of a security role, which is a semantic grouping of operations that a principal is permitted to perform. A method permission is a mapping between one or more security roles and one or more methods that a member of the role can invoke. Container Transactions specify the transaction attributes.
- ▶ **Files** contains the standard 1.1 EJB files and the standard deployment descriptors.

AAT: Entity Bean General Properties

Entity Bean Properties

General | J2EE Extensions | Bindings

An entity bean encapsulates permanent data, which is stored in a data source such as a database or a file system, and associated methods to manipulate that data.

EJB name: *Account

Display name: th CMP to model simple bank accounts

Description:

Home interface: *mples.AccountAndTransfer.AccountHome Browse...

Remote interface: *eSamples.AccountAndTransfer.Account Browse...

EJB class: *mples.AccountAndTransfer.AccountBean Browse...

Primary key class: *mples.AccountAndTransfer.AccountKey Browse...

Primary key field: Compound key

Persistency: Container managed

Reentrant

OK Cancel Help



EJB Name (Required String)

- Specifies a logical name for the enterprise beans.

Display Name

- Specifies a short name to be displayed by the tools.

Description

- Contains text describing the bean

Home Interface Name (Required String)

- Specifies the full name of the enterprise bean's home interface class.

Remote Interface Name (Required String)

- Specifies the full name of the enterprise bean's remote interface class.

EJB Class (Required String)

- Specifies the full name of the bean class

Primary Key Class (Required String)

- Specifies the full name of the bean's primary key class.

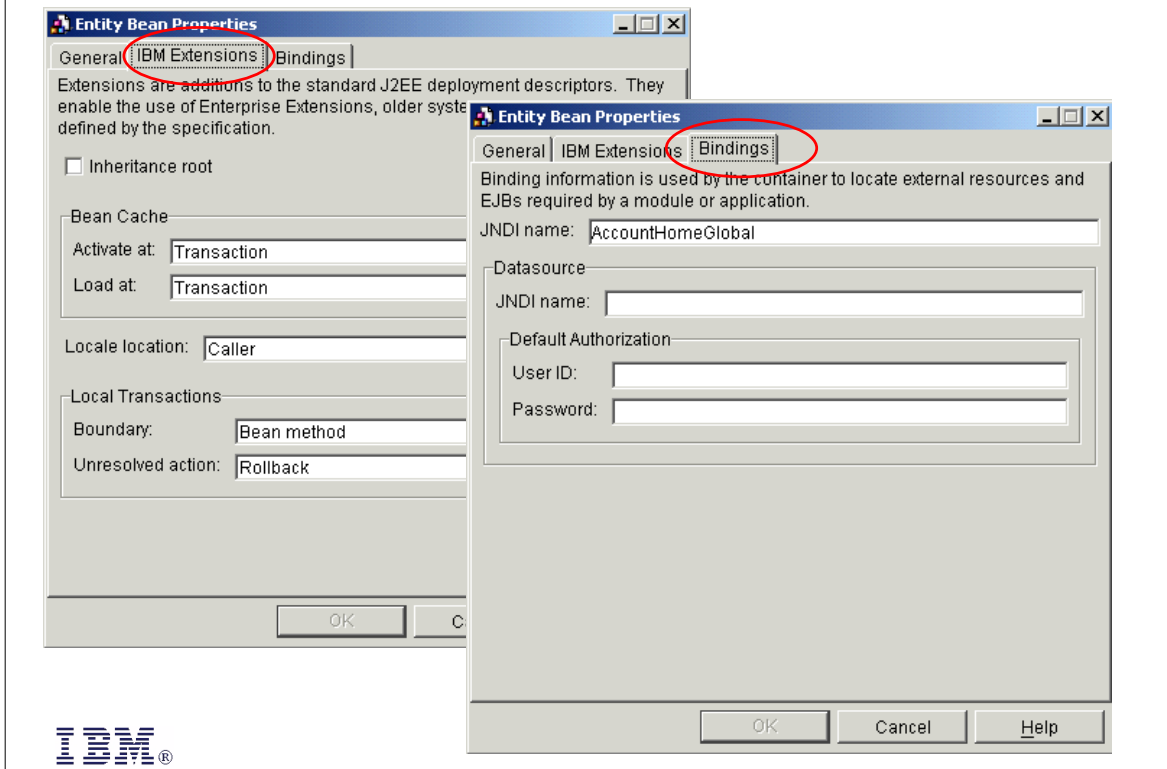
Small Icon

- Specifies the location of a JPEG or GIF file containing a small image(16*16)

Large Icon

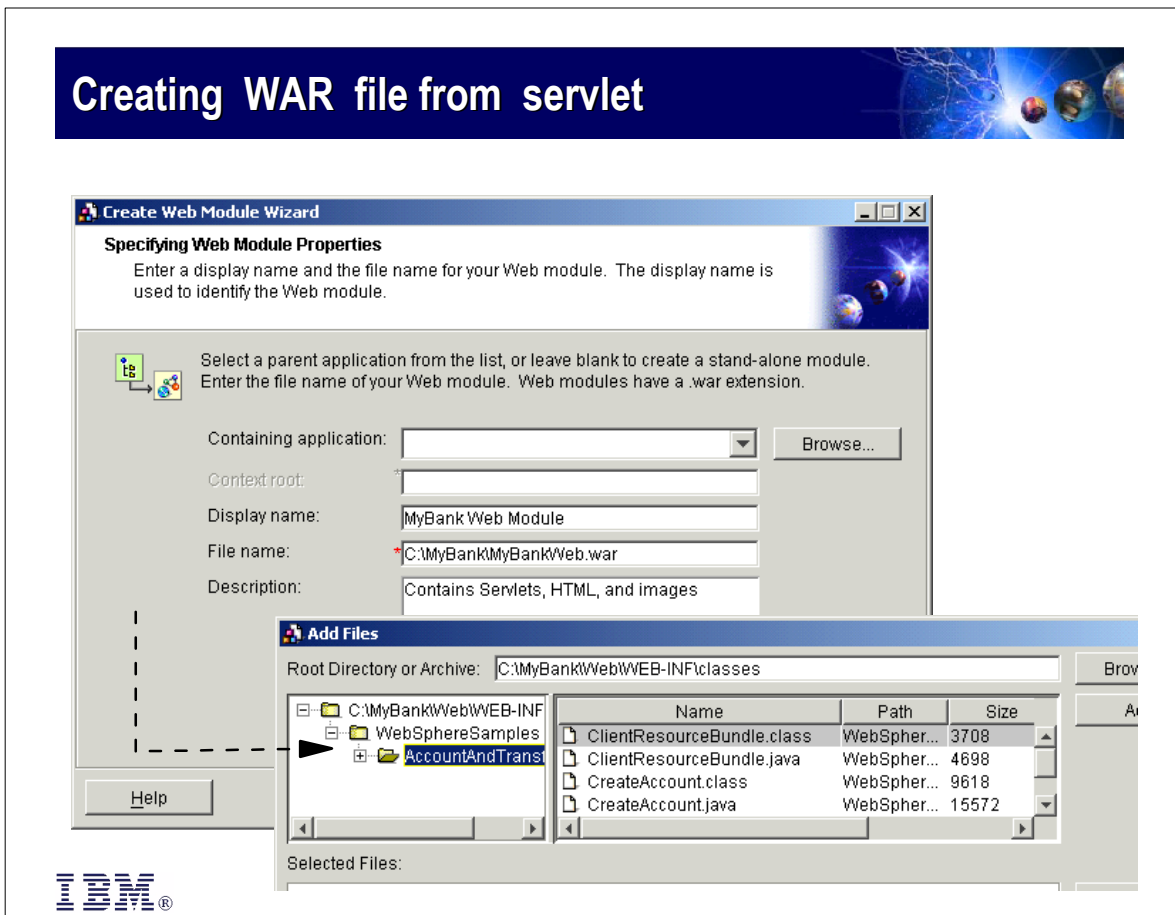
- Specifies the location of a JPEG or GIF file containing a large image (32*32)

AAT: Bean Extensions and Bindings



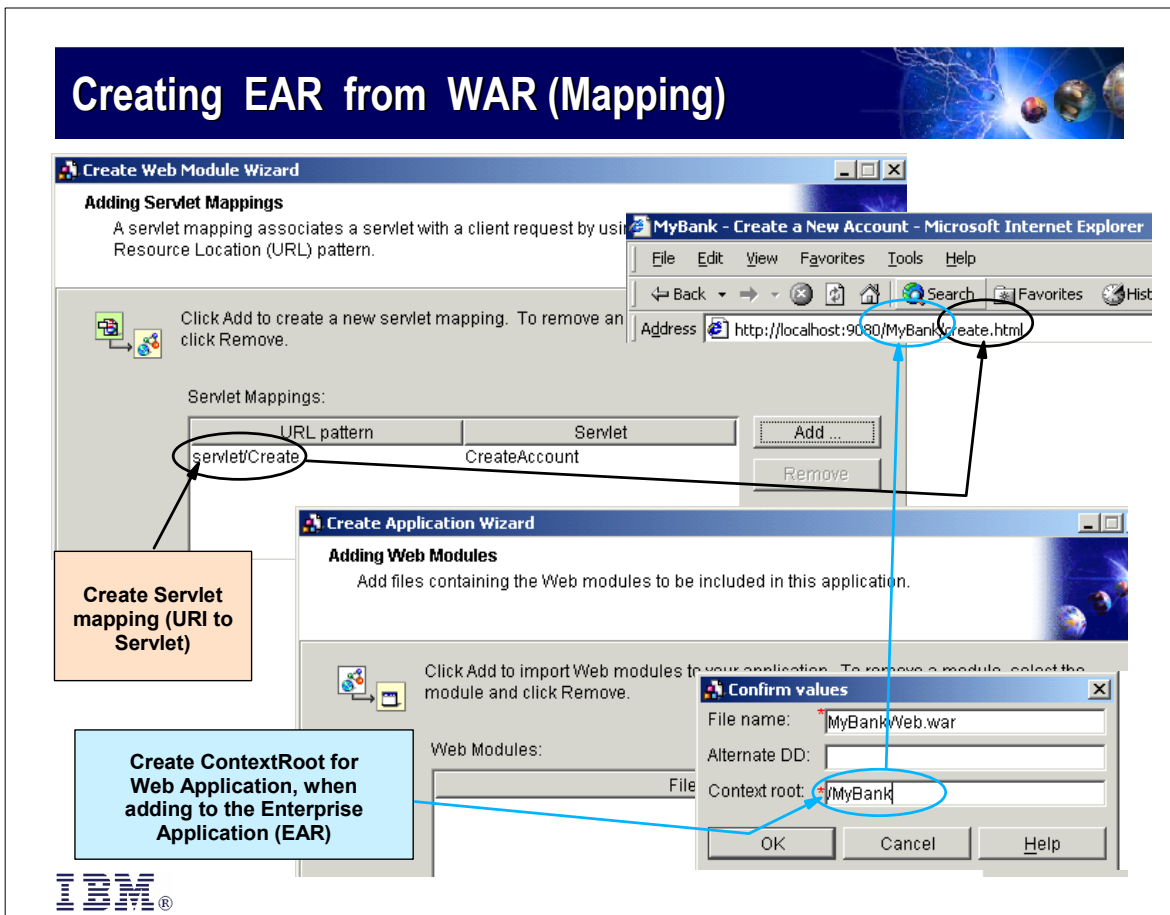
- ▶ Extensions are additions to the standard J2EE deployment descriptors. They enable the use of Enterprise Extensions, older systems, or behavior not defined by the specification.
- ▶ JNDI Name: Specifies the JNDI name of the bean home interface. This is the name under which the enterprise bean home interface is registered and therefore is the name that must be specified when an EJB client does a lookup of the home interface.
- ▶ Data Source JNDI name: Specifies the JNDI name for the bean's data source.

Creating WAR file from servlet



- ▶ Start AAT to bring up the default assembly tool view.
- ▶ Create a WAR file using the wizard available from the tool. Click File --> Wizards --> Create Web Module Wizard.
- ▶ In the Specifying Web Module Properties window, you can type the display name, file name, and description of the Web module.
- ▶ Click Next. In the Adding Files window, click Add Class Files to add the servlets used by the module. You can also add resource files and JAR files.

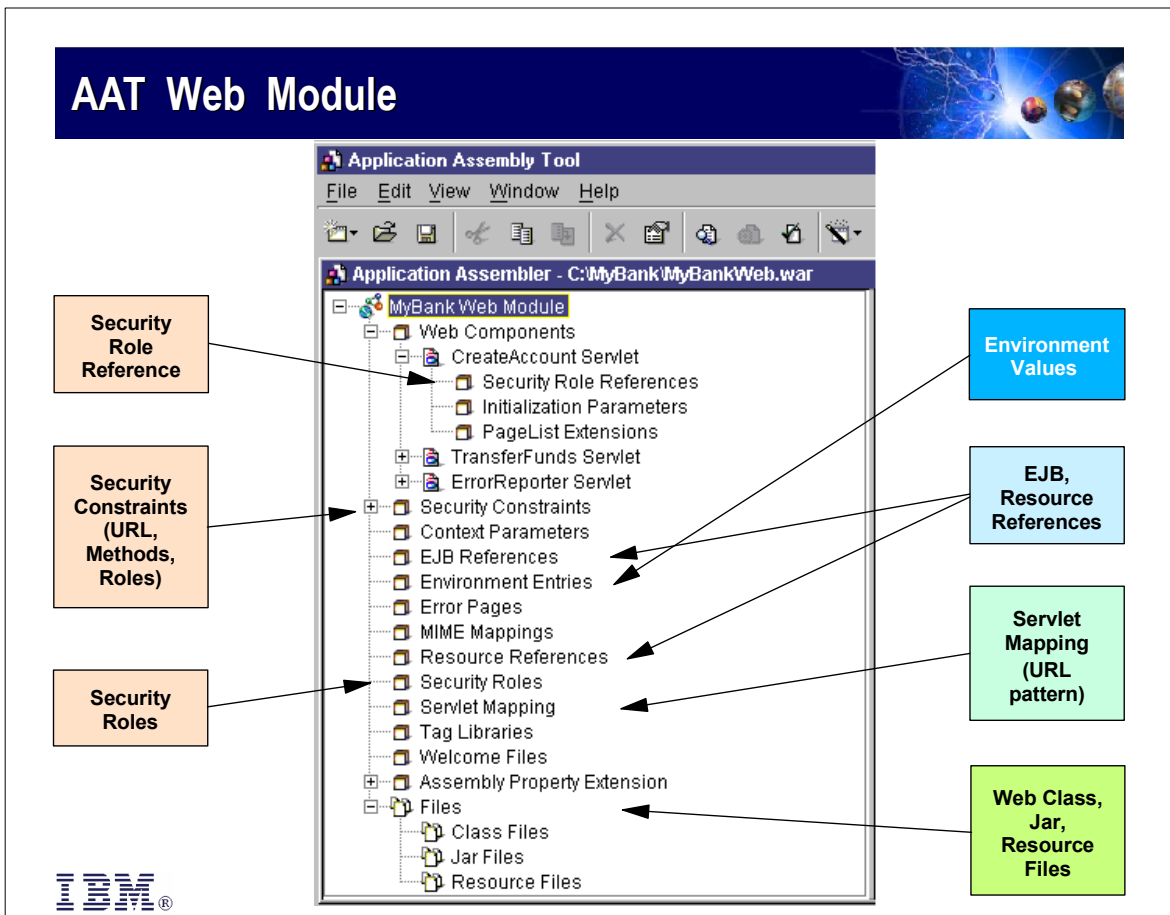
Creating EAR from WAR (Mapping)



Steps to create a EAR file from WAR

- ▶ Use the wizard. Click File --> Wizards --> Create Application Wizard.
- ▶ Follow the directions in the wizard to identify the modules to include, the servlet mappings, and other application-specific information.

AAT Web Module



- ▶ In the Application Assembly Tool, the Web Module default_app has been expanded. default_app contains four Web components - snoop, hello, ErrorReporter and invoker.
- ▶ Just like for the EJB Module shown previously, Environment Values, EJB References, Resource References, Security Role Reference and Security Roles can be specified for the Web Module too.
- ▶ Security Constraints declare how Web content is to be protected. These properties associate security constraints with one or more Web resource collections. A servlet mapping is a correspondence between a client request and a servlet. This is used to specify the URL pattern of the mapping. The Files folder lists general information about the class and resource files.

J2EE Application Client



- Steps to create a J2EE Application client program:
 - ▶ Write the client application program
 - ▶ Assemble the Application Client using AAT
 - ▶ Add the assembled client to an enterprise application
 - ▶ Install the enterprise application on an J2EE Application Server
 - ▶ Run the client in the J2EE Client container (provided by the Application Server). The client container:
 - Provides java:comp/env name space
 - Insulates application from vendor-specific details (e.g. ujc.jar)
 - Locates & launches application main



- ▶ A J2EE application client program is similar to a standard Java program. It runs in its own Java virtual machine and is invoked at its main method. The J2EE application client program differs from a standard Java program because it uses the JNDI name space to access resources. In a Java program, the resource information is coded in the program.
- ▶ Storing the resource information separately from the client application program makes the client application program portable and more flexible.

AAT Client Module

Application Assembly Tool

File Edit View Window Help

Application Assembler - C:\labs\MyBankApp.ea

- MyBank Application
 - EJB Modules
 - Web Modules
 - MyBank Web
 - Application Clients
 - MyBank Client
 - EJB References
 - Resource References
 - Environment Entries
 - Files
 - Security Roles
 - Files

General | Icons

An application client is a standalone Java program.

File name: MyBankClient.jar

Alternate DD:

Display name: MyBank Client

Description:

Classpath: MyBankEJB.jar

Main class: WebSphereSamples.AccountAndTransfer.TransferApplication

Dependent Classpath

Client Container calls this class

EJB, Resource References

Environment Values

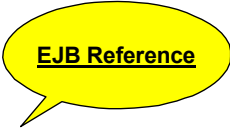
Client Classes, DD

IBM®

J2EE Client - JNDI

```
import javax.naming.*

public class myAppClient
{
    public static void main(String argv[])
    {
        InitialContext ctx = new InitialContext();
        Object myObj = ctx.lookup("java:comp/env/ejb/HelloBean");
        ...
    }
}
```



Using the `javax.naming.InitialContext` class, the client application program uses the lookup operation to access the JNDI name space.



- ▶ The `InitialContext` class provides the lookup method to locate resources.
- ▶ In this example, the program is looking up an EJB called `HelloBean`. The `HelloBean` EJB reference is located in the client JNDI name space at `java:comp/env/ejb/HelloBean`. Since the actual EJB is running on the server, the application client runtime returns a reference to `HelloBean`'s home interface.
- ▶ If the client application program's lookup was for a Resource reference or an Environment entry, then lookup would return an instance of the configured type. For example, if the program's lookup was a JDBC datasource, the lookup would return an instance of `javax.sql.DataSource`.

launchClient tool



- launchClient starts the Application Client runtime.
- launchClient [<userapp.ear> | -help | -?]
[-CC<name>=<value>] [app args]
 - ▶ userapp.ear = The path/name of the .ear file containing the client application.
 - ▶ -help, -? = print this help message.
 - ▶ -CC properties are for use by the Client Container.
 - ▶ "app args" are for use by the client application and are ignored by WebSphere.



- ▶ You have to pass parameters to the launchClient command. You may want to pass parameters to your client application program as well. The launchClient command allows you to do both.
 - ▶ The launchClient command requires the first parameter to be either: an .ear file specifying the client application to launch or a request for launchClient usage information.
 - ▶ All other parameters intended for the launchClient command must begin with the -CC prefix. Parameters that are not .ear files, usage requests, or that begin with the -CC prefix, are ignored by the application client runtime, and are passed directly to the client application program
- CC properties are for use by the Client Container:
- CCverbose = <true|false> Use this option to display additional informational messages.
 - CCjar = The path/name of the JAR file within the EAR file that contains the application you wish to launch. This argument is only necessary when you have multiple client application JAR files in your EAR file.
 - CCBootstrapHost = The name of the host server you wish to connect to initially.
 - CCBootstrapPort = The server port number to use.

IBM WebSphere Application Server V4.0

EJBDeploy Tool



Deploying EJBs : Three ways



- Using AAT, using the "Generate code for deployment" option
- When installing an Enterprise Application
- Using the EJBDeploy command line tool
 - ▶ Used to deploy code in batch mode



- ▶ Before you can successfully run your enterprise beans on either a test or production server, you need to generate deployment code for the enterprise beans.
- ▶ If you want to generate deployment code in batch mode from the command line, you can use the EJBDeploy command.
- ▶ The EJBDeploy command accepts an input EJB JAR file that contains one or more enterprise beans; it then creates an output deployed JAR file that contains generated deployment code for the beans.
- ▶ The command generates .class files and adds them to the deployed JAR file.

EJBDeploy - parameters

ejbdeploy

input_JAR_name

working_directory

output_JAR_name

[-codegen]

[-cp classpath]

[-noinform]

[-novalidate]

[-nowarn]

[-quiet]

[-rmic options]

[-trace]

Only
generate
code - do
not compile
or run RMIC

Additional
jar files

Additional
RMIC
options to
specify

Used for
internal
tracing



input_JAR_name: The fully qualified name of the input JAR file that contains the enterprise beans for which you want to generate deployment code. If your input JAR file contains CMP beans, the `ejbdeploy` command automatically creates the schemas and mappings for the CMP beans.

working_directory: The name of the directory where temporary files are stored that are required for code generation.

output_JAR_name: The fully qualified name of the output JAR file that is created by the `ejbdeploy` command and that contains the generated classes required for deployment.

-codegen: Restricts the `ejbdeploy` command to just (a) importing code from the input JAR file (b) generating the deployment code, and (c) exporting code to the output JAR file. (If you do not specify the `-codegen` option, the `ejbdeploy` command performs the same operations as those performed using the `-codegen` option, but it will additionally compile the generated deployment code and run RMIC.)

-cp classpath: If you intend to run the `ejbdeploy` command against JAR files that have dependencies on other JAR files, you can use the `-cp` option to specify the classpath of the other JAR files.

-noinform: Suppresses informational messages

-novalidate: Prevents the installed validators from running.

-nowarn: Suppresses warning and informational messages.

-quiet: Suppresses status messages (but does not suppress error messages).

-rmic options: Enables you to pass RMIC options to RMIC. (Remote Method Invocation Compiler)

-trace: Generates trace information.

IBM WebSphere Application Server V4.0

EARExpander Tool



EARExpander Tool



- EARExpander expands .ear files into the format desired by the server runtime.
- It can also collapse expanded format to a normal .ear (or .jar, or .zip) format.
- Invocation:
 - ▶ %WAS_HOME%\bin\EARExpander
- Example:
 - ▶ EARExpander -ear my.ear -expandDir c:\tmp\myear -operation expand
 - The above command will expand my.ear into directory c:\tmp\myear
 - Using the -operation collapse would go the other way



IBM WebSphere Application Server V4.0

Installing an Application in WebSphere 4.0



Installing Enterprise Application



- WebSphere 4.0 AEs:
 - ▶ Use the browser-based Admin Console
 - ▶ From command line tool, SEAppInstall

- WebSphere 4.0 AE:
 - ▶ Use the Standalone Java client Admin Console
 - ▶ XMLConfig
 - ▶ WebSphere Control Program (wscp)



AEs: Application Installation



- Steps to install an Enterprise application:
 - ▶ Make sure your server is running
 - Command line: %WAS_ROOT%\bin\startserver or startserverbasic
 - Windows Start --> Programs --> IBM WebSphere --> Application Server V4.0 --> Start Application Server
 - ▶ Start the browser-based admin console
 - <http://localhost:9090/admin>
 - Windows Start --> Programs --> IBM WebSphere --> Application Server V4.0 --> Administrator's Console
 - ▶ Expand Nodes --> Node name
 - ▶ Select Enterprise Applications
 - ▶ Click Install
 - ▶ Browse and open the Enterprise archive (.ear) file to install
 - ▶ Enter Virtual Host Name
 - ▶ Save the new configuration
 - Updates the %WAS_ROOT%\config\server-cfg.xml file



AEs: Command Line App Installer



- The SEAppInstall tool takes an assembled J2EE application (.ear file) and prepares it to run in AEs.
 - ▶ Walks the modules in the .ear file to locate & resolve bindings
 - ▶ Updates server-cfg.xml file to reflect that the application is installed
 - ▶ Expands the .ear file into the desired format for the server run time
 - ▶ Located in %WAS_ROOT%/bin/SEAppInstall



- ▶ Although they can be installed anywhere, application files are typically put in the product_installation_root/installedApps directory. This is the default directory in which the application installer places files.
- ▶ Installed J2EE applications (.ear files) are exploded into a directory having the same name as the original .ear file.
 - ▶ This makes it easier for the runtime class loader. You will notice that some J2EE modules (.jar files and .war files) might also be exploded as subdirectories.
 - ▶ You might also notice some .xmi files, which are remnants of the application assembly, deployment and installation processes.
 - ▶ These are primarily the product-specific bindings and extensions.

AEs: SEAppInstall - Tasks



- Install an application into a server
- Uninstall an application from a server
- Export an installed application containing configuration information from a server
- List displays all installed Enterprise applications



AEs: SEAppInstall Tool -install



- The -install option will take an EAR file and configure it within a server config file
- The -install option contains 4 optional parameters:
 - ▶ -expandDir
 - ▶ -nodeName
 - ▶ -serverName
 - ▶ -interactive
- Expand the application under the expandDir location, if specified
 - ▶ The expanded directory will be named the same name as the installed application

```
SEAppinstall -install %WAS_ROOT%\installableApps\sampleApp.ear  
-configFile %WAS_ROOT%\config\server-cfg.xml -interactive false
```



- ▶ The -install option will take an EAR file (or expanded directory containing the EAR file contents) and configure it within a server config file (specified via the -configFile parameter).
- ▶ The example will install the sampleApp.ear file into appropriate server configuration file.

SEAppInstall - Uninstall an Application



- The -uninstall option will take an EAR file and remove it from a server config file
- The location of the installed application in the expanded directory will not be removed.

```
seappinstall -uninstall sampleApp -configFile  
%WASROOT%\config\server-cfg.xml
```

Note that sampleApp is the <display-name> attribute from the .ear file's deployment descriptor.

- The -uninstall option contains two optional parameters:
 - ▶ -nodeName
 - ▶ -serverName



- ▶ The -uninstall option will take an EAR file and remove it within a server config file (specified via the -configFile parameter). Note: the location of the installed application in the expanded directory will not be removed

The process of uninstalling an application will update the server configuration file removing the application. It will remove the module references from being installed on the EJB and Web Containers. For example:

```
seappinstall -uninstall sampleApp -configFile  
c:\WebSphere\SingleServer\config\server-cfg.xml
```

This will uninstall the sampleApp application from the appropriate server configuration file. Note that sampleApp is the <display-name> attribute from the .ear file's deployment descriptor.

SEAppInstall - Export an application



- The -export option will take installed application information within a server config file and export it to the location specified.
- Command Example

```
SEAppinstall -export sampleApp -configFile  
              %WAS_ROOT%\config\server-cfg.xml -outputFile  
              c:\exportedApps\sampleApp.ear
```



- ▶ The -export option will take an installed application within a server config file (specified via the -configFile parameter) and export it to the location specified by the -outputFile parameter. This is useful because the exported jar will contain the binding information that the application used within the server configuration. For example:

```
seappinstall -export sampleApp -configFile  
c:\WebSphere\SingleServer\config\server-cfg.xml -outputFile  
c:\exportedApps\sampleApp.ear
```

This will export the sampleApp application from the appropriate server configuration file and place the configuration information in the file specified by the outputFile parameter

SEAppInstall - List installed configuration



- The -list option will list the following:
 - ▶ Enterprise Applications
 - ▶ Web Applications
 - ▶ EJBs
 - ▶ All of the above
- Parameters:
 - ▶ apps
 - ▶ wars
 - ▶ ejbjars
 - ▶ all
- Command example:
`seappinstall -list all -configFile %WAS_ROOT%\config\server-cfg.xml`



Lab Preview - Build an Enterprise Application

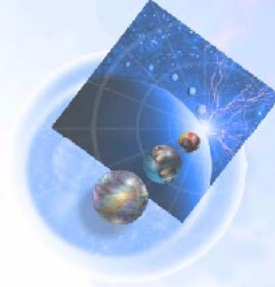


- Put on the "Application Assembler" hat
- Build an EJB Jar file from provided classes
 - ▶ "Gotcha" alert - Don't use the "Back" button in this lab.
- Build a Web Module from provided servlets, HTML, and image files
- Build a Client Module from provided classes
- Assemble the WAR and JAR files into an EAR file

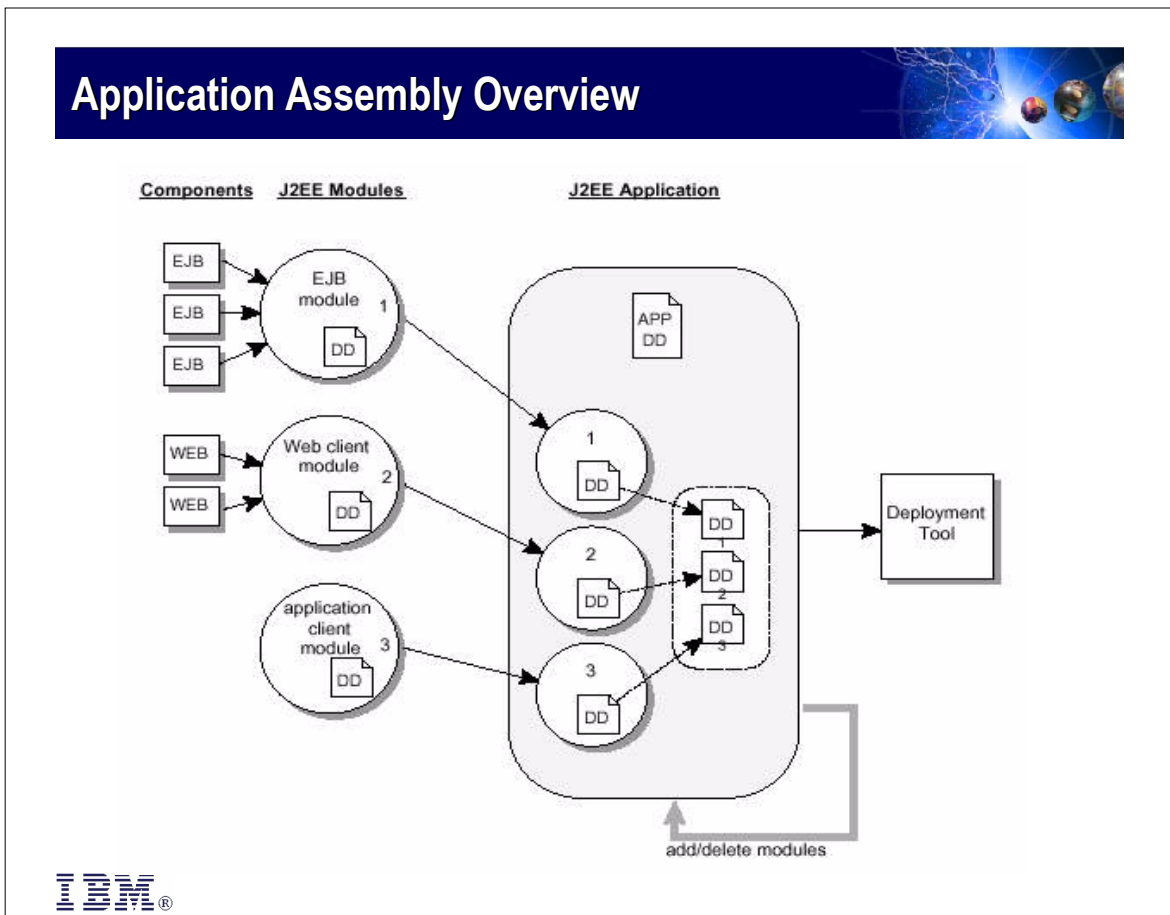




Appendix: Deployment Descriptors

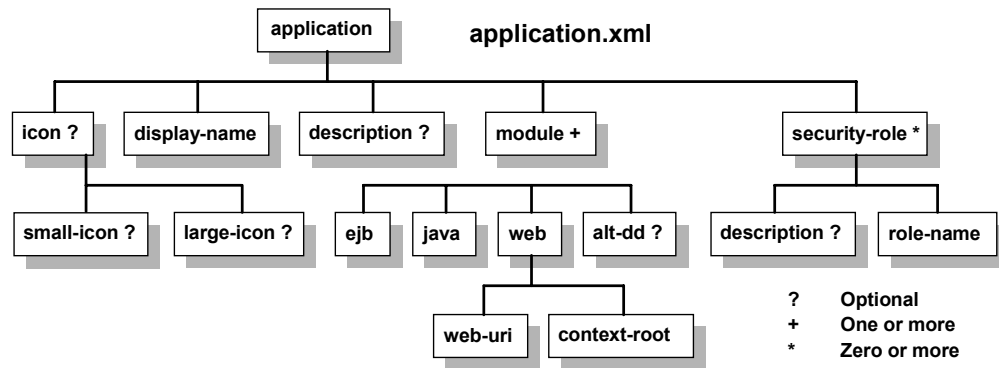


Application Assembly Overview



- ▶ J2EE applications are composed of one or more J2EE components and one J2EE application deployment descriptor.
- ▶ The deployment descriptor lists the application components as modules.
- ▶ A J2EE module represents the basic unit of composition of a J2EE application.
- ▶ J2EE modules consist of one or more J2EE components and one component level deployment descriptor.
- ▶ The flexibility and extensibility of the J2EE component model facilitates the packaging and deployment of J2EE components as individual components, component libraries, or J2EE applications.
- ▶ EJB Modules contain one or more EJB components.
- ▶ Web Client Application Modules contain one or more of Web components (this could be servlets, JSPs, or both).
- ▶ Application Client Modules contain an application client.
- ▶ Each Module also includes its own deployment descriptor (DD). The Application that contains the modules have its own DD. Each deployment descriptor is specified by the corresponding XML Document Type Definition (DTD).

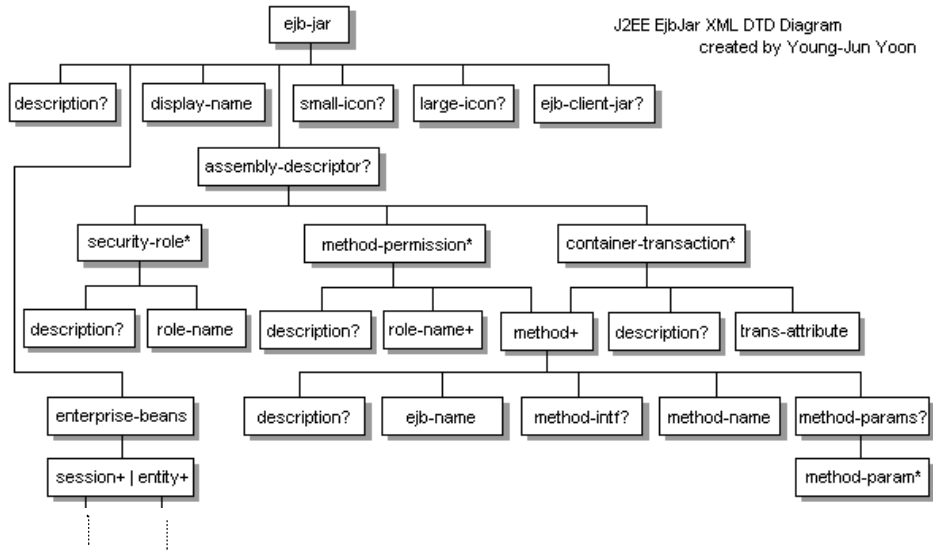
J2EE Application XML DTD



DTD defines the XML grammar for a J2EE application deployment descriptor:

- ▶ The **application** element is the root element of a J2EE application deployment descriptor.
- ▶ The **icon** element (`small-icon?`, `large-icon?`) contains a `small-icon` and `large-icon` element that specifies the URIs for a small and a large GIF or JPEG icon image to represent the application in a GUI.
- ▶ The **display-name** element specifies an application name. The application name is assigned to the application by the application assembler and is used to identify the application to the deployer at deployment time.
- ▶ The **description** element provides a human readable description of the application. The description element should include any information that the application assembler wants to provide the deployer.
- ▶ The **module** element (`(ejb | java | web)`, `alt-dd ?`) represents a single J2EE module and contains an EJB, Java, or Web element that indicates the module type, contains a path to the module file, and an optional `alt-dd` element that specifies an optional URI to the post-assembly version of the deployment descriptor. The application deployment descriptor must have one module element for each J2EE module in the application package.
 - ▶ The **ejb** element specifies the URI of an EJB JAR file, relative to the top level of the application package.
 - ▶ The **java** element specifies the URI of a Java application client module, relative to the top level of the application.
 - ▶ The **web** element contains the Web URI and `context-root` of a Web application module.
 - ▶ The **web-uri** element specifies the URI of a Web application file, relative to the top level of the application.
 - ▶ The **context-root** element specifies the context root of a web application
 - ▶ The **alt-dd** element specifies an optional URI to the post-assembly version of the deployment descriptor file for a particular J2EE module. The URI must specify the full pathname of the deployment descriptor file relative to the application root directory. If `alt-dd` is not specified, the deployer must read the deployment descriptor from the default location and file name required by the respective component specification.
- ▶ The **security-role** element (`description?`, `role-name`) contains the definition of a security role that is global to the application. The definition consists of a description of the security role, and the security role name. The descriptions at this level override those in the component level security-role definitions and must be the descriptions tool display to the deployer.
 - ▶ The **role-name** element contains the name of a security role.

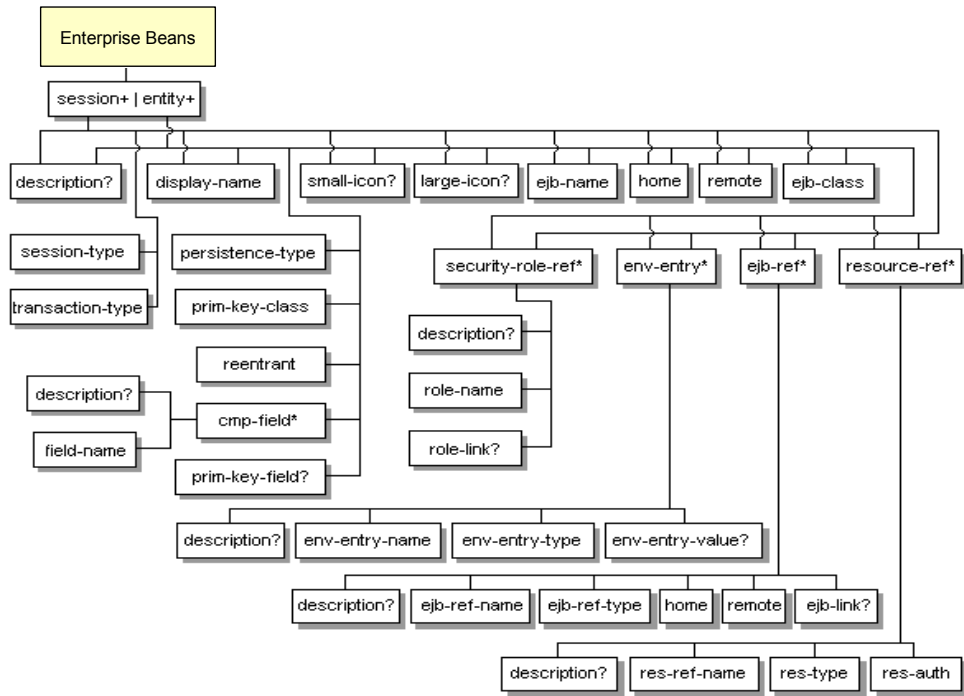
J2EE Enterprise JavaBean XML DTD (1-2)



cont.. on next page



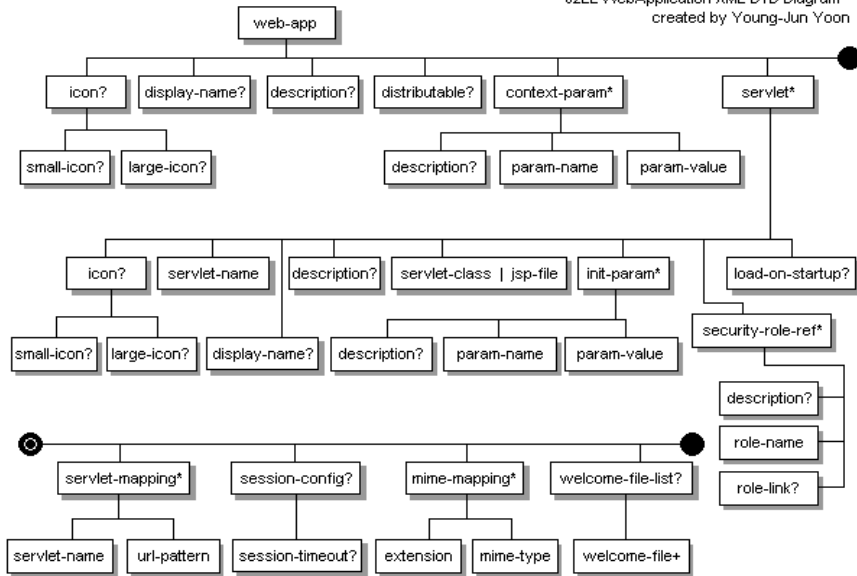
J2EE Enterprise JavaBean XML DTD(2-2)



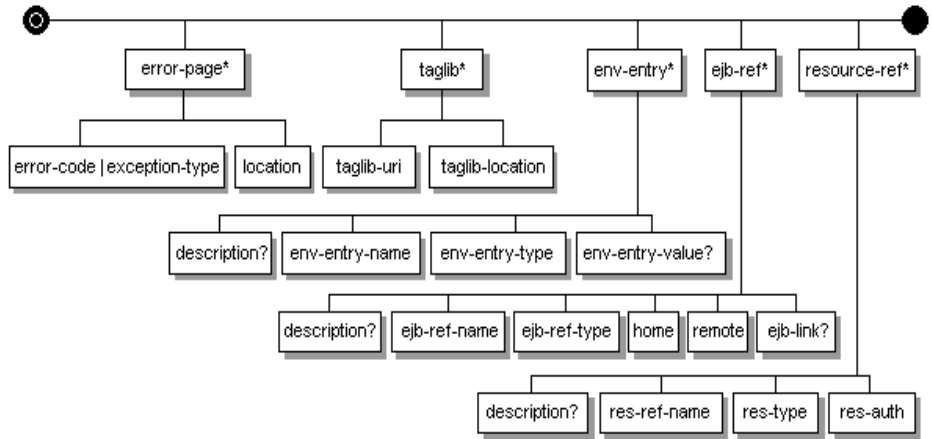
J2EE Web Application XML DTD (1-3)



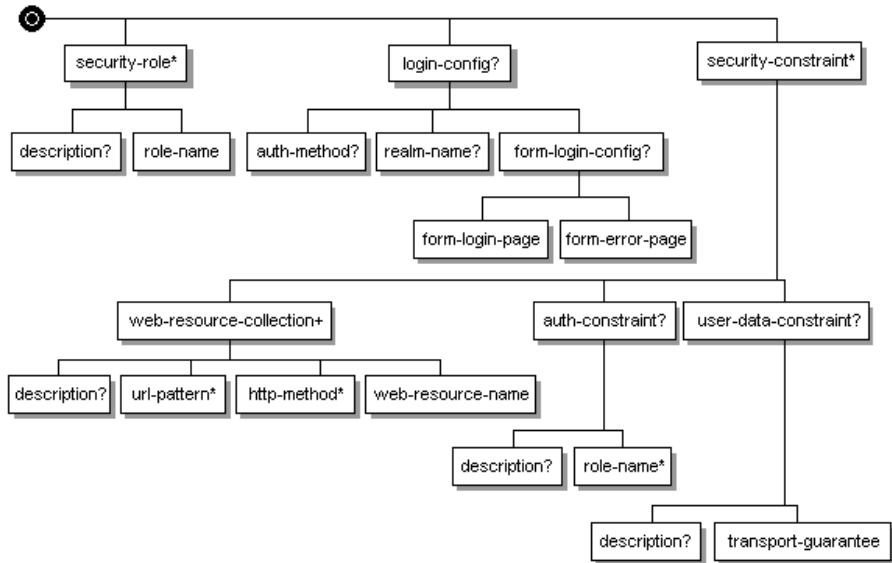
J2EE WebApplication XML DTD Diagram
created by Young-Jun Yoon



J2EE Web Application XML DTD (2-3)



J2EE Web Application XML DTD (3-3)



J2EE Application Client XML DTD

