# Enterprise beans -- table of contents

## Development



## Administration

# 4.3: Developing enterprise beans

Enterprise applications are applications that typically use enterprise beans. To develop enterprise applications, you must:

1. Develop any session or entity beans your application will use
2. Create the deployment descriptor and the EJB JAR file.
3. Deploy the enterprise beans.

Enterprise applications support both transactions and security.

*Writing Enterprise Beans* is a programming guide for developing, packaging, anddeploying enterprise beans in IBM WebSphere Application Server. It discussesboth the Advanced Edition and Enterprise Edition of the product.

| Format |
| --- |
| PDF |
| HTML |

See section 4.3.1 for additional information that could not be added to the book in time for this product release.

# 4.3.1: Late-breaking enterprise beans programming tips

This article provides programming tips and considerations to supplementthe Writing Enterprise Beans book. Also see the product Release Notes.

## EJB jar files that contain both source and class files result in compile errors or exceptions

Ensure your jar files only contain class files, images, and sounds. Source (`.java`) files in your jar file will cause exceptions when you run the application, or compile errorswhen you compile the source.

## Disregard README.rmi-iiop / README.RMI-IIOP

The product installs an unnecessary file:

*`product_installation_root`*`/java/README.RMI-IIOP`

If you encounter this file, disregard it. It instructs you to rename an rmictools.jar file that does not exist. Because the wstools.jar is already installed,containing the necessary IBM implementations for IIOP, you do not need torename the file in order to use the IBM rmic (Remote Method Invocation) compiler.

## Avoid creating or accessing protected enterprise beans in the servlet init() method

The Writing Enterprise Beans book contains a discussion of servletinit() methods in the context of developing servlets that use enterprise beans. Hereis some additional information about the security aspect.

Although the init method is a good place to get references to EJB home objects, it is not a good place to create enterprise beans or access other enterprise beans that might be protected with WebSphere security. Depending on the authorization policy on the protected objects, creating or accessing these objects from withinthe servlet init() method could fail for authentication or authorization reasons because they were not accessed with the proper security credentials.

Creating or accessing protected objects should be doneafter the init() method, in one of the *doXXX* methods of the servlet.

## Deployment tool limitations

The enterprise bean deployment tool provided by WebSphere Application Servermaps all non-primitive Java types to serialized BLOB objects when the beans are using a DB2database. For example, when a CMP bean with field java.math.BigDecimal is deployedon DB2, its field becomes a BLOB data type.

If you need to map non-primitive types to other, more complex datatypes, consider using IBM VisualAge for Java for deployment.

## Inheritance by remote objects

An enterprise bean or other remote object cannot inheritfrom two interfaces that have methods with the same name, even if those methodshave different signatures, due to the Java-IDL mapping specification.

Java programmers accustomed to the usual Java inheritance model shouldtake care to note this limitation of the specification. By the Enterprise JavaBeans (EJB) specification, enterprise beans should not be written to inherit from two interfaces as described above. If they do, they will encounter errors when deployed.

## Option A caching incompatible with clusters and shared data

When Option A caching is in use, the application server hosting the enterprise bean container must be the only updater of the data in the persistent store. As such, Option A caching is incompatible with:

- Workload managed servers (such as a cluster of clones)
- Database with data being shared among multiple applications

Shared database access corresponds to Option C caching. See theEJB specification for futher details.

Option A and Option C caching are also known as commit option Aand commit option C, respectively.

## Best practice for data source ID and password

Although it is not necessary, it is good practice to specify the user IDand password for a data source either in the enterprise bean to be using the data source, or the container of the bean.

## Developer's Client Files for setting up Java application clients

In "Developing EJB clients," the Writing Enterprise Beans book states:

> The Java client object request broker (ORB), which is automatically initialized in EJB clients, does not support dynamic download of implementation bytecode from the server to the client. As a result, all classes required by the EJB client at runtime must be available from the files and directories identified in the client's CLASSPATH environment variable. For information on the JAR files required by EJB clients, see Setting the CLASSPATH environment variable in the EJB server (AE) environment or Setting the CLASSPATH environment variable in the EJB server (CB) environment.

Article 1.4 about installable components describes how to install the needed files on your machine. See the tableentry for "full Java application client."

Note a possible book correction: The English version of the Writing Enterprise Beans book refers only to one of two installation options presented in article 1.4.1 -- you mightneed the option that it does not mention. The translated versions of thebook do not mention either option for installing a full Java applicationclient.

## Commiting transactions based on EJB 1.1 specification

According to the EJB specifications, if an enterprisebean container catches an exception from the business method of anenterprise bean, and the method is running within a container managed transaction, the container should rollback the transaction before passing the exception on to the client.

However, if the business method is throwing an Application exception as defined in Chapter 12 of EJB 1.1 specification, then the normal behavior for the container in this case is to COMMIT the transaction. Even thoughIBM WebSphere Application Server Version 3.5 does not officially supportthe EJB 1.1. specification level, in such a case it behaves as determinedby the 1.1. specification. If a business method throws an exception, the container will commit the transaction before re-throwing the exception.

## EJB clients need ioser library to run

If using Windows NT, ensure that EJB clients can locate the following library file at their run time: ioser.dll

## References to jar file, iioptools.jar, should be ignored.

The Writing Enterprise Beans book contains many references to file, **iioptools.jar**.These references should be ignored. This was a required jar file for JDK levels priorto JDK 1.2.2, and had to be defined in the CLASSPATH for WebSphere Application Server to executesuccessfully. With JDK 1.2.2, file, **iioptools.jar**, was incorporated into theruntime environment, and no longer needs to be included in the CLASSPATH. In fact, withJDK 1.2.2, file, **iioptools.jar**, no longer exists.

FP 3.5.2

# 4.3.2: JNDI caching

In IBM WebSphere Application Server Advanced Edition,JNDI context objects employ caching in order to increase the performanceof JNDI lookup operations. Objects bound and looked up are cached in orderto speed up subsequent lookups of those objects. Objects are cached asthey are bound or initially looked up. Normally, JNDI clients should beable to simply use the default cache behavior. The following sectionsdescribe in detail cache behavior, and how JNDI clients can override defaultcache behavior if necessary.

- Cache behavior
- Cache properties
- Coding examples

## Cache behavior

A cache is associated withan initial context when a javax.naming.InitialContext object is instantiatedwith the java.naming.factory.initial property set to:

```
com.ibm.ejs.ns.jndi.CNInitialContextFactory
```

CNInitialContextFactory searches the environment properties for a cachename, defaulting to the provider URL. If no provider URL is defined, acache name of "iiop:///" is used. All instances of InitialContext whichuse a cache of a given name share the same cache instance.

After an associationbetween an InitialContext instance and cache is established, the associationdoes not change. A javax.naming.Context object returned from a lookup operationwill inherit the cache association of the Context object on which the lookupwas performed. Changing cache property values with the Context.addToEnvironment()or Context.removeFromEnvironment() method does not affect cache behavior.Properties affecting a given cache instance, however, may be changed witheach InitialContext instantiation.

A cache is restricted toa process and does not persist past the life of the process. A cached objectis returned from lookup operations until either the max cache life forthe cache is reached, or themax entry life for the object's cache entryis reached.

After this time, a lookup on the object will cause the cacheentry for the object to be refreshed. If a bind or rebind operation isexecuted on an object, the change will not be reflected in any caches otherthan the one associated with the context from which the bind or rebindwas issued. This "stale data" scenario is most likely to happen when multipleprocesses are involved, since different processes do not share the samecache, and Context objects in all threads in a process will typically sharethe same cache instance for a given name service provider.

Usually, cached objects are relatively static entities, and objects becoming staleshould not be a problem. However, timeout values can be set on cache entriesor on a cache itself so that cache contents are periodically refreshed.

## Cache properties

JNDI clients can use several properties to control cache behavior.These properties can be set in the JVM system environment or in theenvironment Hashtable passed to the InitialContext constructor.

Cache properties are evaluated when an InitialContext instance iscreated. The resulting cache association, including"none", cannot bechanged. The "max life" cache properties affectthe individual cache's behavior. If the cache already exists, cache behavior will beupdated according to the new "max life" property settings. If no"max life" properties exist in the environment, the cachewill assume default "max life"settings, irrespective of the previous settings.The various cache properties are describedbelow. All property values must be string values.

- **com.ibm.websphere.naming.jndicache.cacheobject**

  Caching is turned on or off with this property. Additionally, an existingcache can be cleared.Listed below are the valid values for this property and the resulting cachebehavior:

  - "**populated**" (default): Use a cache with the specified name. If the cache already exists, leave existing cache entries in cache; otherwise, create a new cache.
  - "**cleared**": Use a cache with the specified name. If the cache already exists, clear all cache entries from cache; otherwise, create a new cache.
  - "**none**": Do not cache. If this option is specified, the cache name is irrelevant. Therefore, this option will not disable a cache that is already associated with other InitialContext instances. The InitialContext being instantiated will not be associated with any cache.

- **com.ibm.websphere.naming.jndicache.cachename**

  It is possible to createmultiple InitialContext instances, each operating on the namespace of adifferent name service provider. By default, objects from each serviceprovider are cached separately, since they each involve independent namespacesand name collisions could occur if they used the same cache. The providerURL specified when the initial context is created serves as the defaultcache name. With this property, a JNDI client can specify a cache nameother than the provider URL. Listed below are the valid options forcache names:

  - "**providerURL**" (default): Use the value for java.naming.provider.url property as the cache name. The default provider URL is "iiop:///". URLs are normalized by stripping off everything after the port. For example, "iiop://server1:900" and "iiop://server1:900/com/ibm/initCtx" are normalized to the same cache name.

- ***Any string***: Use the specified string as the cache name. Any arbitrary string with a value other than "providerURL" can be used as a cache name.

- **com.ibm.websphere.naming.jndicache.maxcachelife**

  By default, cached objects remain in the cache for the life of the process oruntil cleared with the com.ibm.websphere.naming.jndicache.cacheobject propertyset to "cleared". This property enables a JNDI client to set the maximum lifeof a cache as follows:

  - "**0**" (default): Make the cache lifetime unlimited.
  - ***Positive integer***: Set the maximum lifetime of the cache, in minutes, to the specified value. When the maximum cache lifetime is reached, the cache is cleared before another cache operation is performed. The cache is repopulated as bind, rebind, and lookup operations are executed.

- **com.ibm.websphere.naming.jndicache.maxentrylife**

  By default, cached objects remain in the cache for the life of the processor until cleared with the com.ibm.websphere.naming.jndicache.cacheobjectproperty set to "cleared". This property enables a JNDI client to set themaximum lifetime of individual cache entries as follows:

  - "**0**" (default): Lifetime of cache entries is unlimited.
  - ***Positive integer***: Set the maximum lifetime of individual cache entries, in minutes, to the specified value. When the maximum lifetime for an entry is reached, the next attempt to read the entry from the cache will cause the entry to be refreshed.

## Coding examples

```
import java.util.Hashtable;import javax.naming.InitialContext;import javax.naming.Context;/*****
Caching discussed in this section pertains only to the WebSphere Advanced Edition initial context
factory. Assume the property, java.naming.factory.initial, is set to
"com.ibm.ejs.ns.CNInitialContextFactory" as a java.lang.System property.*****/Hashtable env;Context
ctx;// To clear a cache:env = new
Hashtable();env.put("com.ibm.websphere.naming.jndicache.cacheobject", "cleared");ctx = new
InitialContext(env);// To set a cache's maximum cache lifetime to 60 minutes:env = new
Hashtable();env.put("com.ibm.websphere.naming.jndicache.maxcachelife", "60");ctx = new
InitialContext(env);// To turn caching off:env = new
Hashtable();env.put("com.ibm.websphere.naming.jndicache.cacheobject", "none");ctx = new
InitialContext(env);// To use caching and no caching:env = new
Hashtable();env.put("com.ibm.websphere.naming.jndicache.cacheobject", "populated");ctx = new
InitialContext(env);env.put("com.ibm.websphere.naming.jndicache.cacheobject", "none");Context
noCacheCtx = new InitialContext(env);Object o;// Use caching to look up home, since the home should
rarely change.o = ctx.lookup("com/mycom/MyEJBHome");// Narrow, etc. ...// Do not use cache if data
is volatile.o = noCacheCtx.lookup("com/mycom/VolatileObject");// ...
```

FP
3.5.3

# 4.3.3: Using Java Message Service (JMS) resources

WebSphere Application Server Enterprise JavaBeans now support the transactional use of MQSeries Java Message Service (JMS) resources.

To use this feature, install MQSeries version 5.2 and the MQSeries classes for Java and JMS.Only MQSeries V5.2 provides this support; earlier versions will not work.

To configure JMS resources for use with WebSphere Application Server:

1. Download the MQSeries Java and JMS classesfrom URL, http://www.ibm.com/software/ts/mqseries/api/mqjava.html
2. Review the *MQSeries Using Java* book which describes how to configure JMS resourcesfor use with WebSphere Application Server.
3. Use the MQSeries administration tool, JMSAdmin, to bind the Java and JMS classes to the JNDI namespace.
4. Configure the following three parameters of the MQSeries administration tool, JMSAdminto support WebSphere Application Server:
    - INITIAL_CONTEXT_FACTORY
    - PROVIDER_URL
    - SECURITY_AUTHENTICATION
5. Review the WebSphere Application Server specific configuration instructions.
6. Review the WebSphere Application Server JMS connection factories in JMSAdmin,specifically:
    - **WSQCF** - queue connection factory
    - **WSTCF** - topic connection factory

## WebSphere Application Server connection factory objects

All ***QueueSession*** and ***TopicSession*** objects created from the WebSphereApplication Server connection factories are *transacted*, and require an active transaction for the following calls:

- QueueSender.send
- MessageConsumer.receive
- MessageConsumer.receiveNoWait
- TopicPublisher.publish

1. Using these calls in an unspecified transaction context, that iswhen there is ***no*** active transaction, is ***not*** supported.
2. Messages sent via *QueueSender.send* or published using *TopicPublisher.publish* do not become visibleuntil the transaction is committed.
3. Messages received by *MessageConsumer.receive* or *MessageConsumer.receiveNoWait* are requeued if the transaction is rolled back.
4. Both bean-managed transaction demarcation and container-managed demarcation are

supported.

5. Calls to *QueueConnection.createQueueSession* and *TopicConnection.createTopicSession*are given the parameters:
   - ***true*** (transacted)
   - ***0*** (zero, since *acknowledgeMode* is not relevant).

   The actual values of the parameters are ignored.
6. You ***cannot*** obtain a non-transacted session from the WebSphere Application Server JMS connection factories. To create a non-transacted session, you mustuse a conventional queue connection factory or topic connection factory such as **QCF** or **TCF** in the JMSAdmin tool.
7. Requestors are only used with non-transacted sessions. Therefore, *QueueRequestor*and *TopicRequestor* cannot be used with sessions created by WebSphere ApplicationServer JMS connection factories.
8. With the Enterprise JavaBeans programming model, you must ensure all JMS resources are closed correctly.Since JMS resources never time-out, JMS resources that are not closed correctly will continue to consume MQSeries resources.The MQSeries resources also persist until the application server or MQSeries Queue manager is restarted.

## Unsupported interfaces and methods

The following JMS interfaces are not designed for application use and, therefore,cannot be invoked:

| Unsupported interfaces |
|---|
| *javax.jms.ServerSession* |
| *javax.jms.ServerSessionPool* |
| *javax.jms.ConnectionConsumer* |
| all the *javax.jms.XA* interfaces |

The following JMS methods are inappropriatein this environment and interfere with connection management by the container. Therefore, these methods cannot be used:

| Unsupported methods |
|---|
| *javax.jms.Connection.setExceptionListener* |
| *javax.jms.Connection.stop* |
| *javax.jms.Connection.setClientID* |
| *javax.jms.Connection.setMessageListener* |
| *javax.jms.Session.getMesssageListener* |
| *javax.jms.Session.run* |
| *javax.jms.QueueConnection.createConnectionConsumer* |
| *javax.jms.TopicConnection.createConnectionConsumer* |
| *javax.jms.TopicConnection.createDurableConnectionConsumer* |
| *javax.jms.MessageConsumer.setMessageListener* |
| *javax.jms.Session.commit* |
| *javax.jms.Session.rollback* |
| *javax.jms.Session.recover* |
| *javax.jms.Message.acknowledge* |

You cannot register a *MessageListener*with a *QueueReceiver* or *TopicSubscriber*.These restrictions match the onesdocumented in the Enterprise JavaBeans 2.0 specification.

**Related information...**

- 4.3: Developing enterprise beans
- InfoCenter (productdocumentation)

# About this book

This document focuses on the development of enterprise beans written to theSun Microsystems Enterprise JavaBeans(TM) specification in the WebSphere(TM)Application Server programming environment. It also discussesdevelopment of EJB clients that can access enterprise beans.

---

## Who should read this book

This document is written for developers and system architects who want anintroduction to programming enterprise beans and EJB clients in WebSphereApplication Server. It is assumed that programmers are familiar withthe concepts of object-oriented programming, distributed programming, andWeb-based programming. Knowledge of the Sun Microsystems Java(TM)programming language is also assumed.

---

## Document organization

This document is organized as follows:

- An architectural overview of the EJB programming environment provides a high-level introduction to the EJB serverenvironment in WebSphere Application Server.
- An introduction to enterprise beans explains the main concepts associated with enterprisebeans.
- Tools for developing and deploying enterprise beans in the EJB server (AE) environment explains how to set up and use the tools contained in theEJB server (AE) environment. It also discusses the major steps indeveloping and deploying enterprise beans in that environment. The EJBserver (AE) is the EJB server implementation available with the WebSphereApplication Server Advanced Edition.
- Tools for developing and deploying enterprise beans in the EJB server (CB) environment explains how to set up and use the tools contained in theEJB server (CB) environment. It also discusses the major steps indeveloping and deploying enterprise beans in that environment. The EJBserver (CB) is the EJB server implementation available with Component Brokeras part of the WebSphere Application Server Enterprise Edition.
- Developing enterprise beans explains how to develop entity beans with container-managedpersistence (CMP) and session beans. It also provides information onhow to package enterprise beans for later deployment.
- Enabling transactions and security in enterprise beans explains how to enable transactions in enterprise beans byusing the appropriate deployment descriptor attributes.
- Developing EJB clients explains the basic code required by an EJB client to use anenterprise bean. This chapter covers generic issues relevant toenterprise beans, Java applications, and Java servlets that use enterprisebeans.
- Developing servlets that use enterprise beans discusses the basic code required in a servlet that accessesan enterprise bean.
- More-advanced programming concepts for enterprise beans explains how to develop a simple entity bean withbean-managed persistence and discusses the basic code required of anenterprise bean that manages its own transactions.
- Appendix A, Changes for version 1.1 of the EJB specification describes features that are new or have changed in version1.1 of the EJB specification and discusses migration issues forenterprise beans written to version 1.0 of the EJBspecification.
- Appendix B, Example code provided with WebSphere Application Server describes the major example

used throughout this book andthe additional examples that are delivered with the various editions ofWebSphere Application Server.

- Appendix C, Using XML in enterprise beans (CB Only) describes the extensible markup language (XML) that can beused to create deployment descriptors for use with enterprise beans in the EJBserver (CB) environment.
- Appendix D, Extensions to the EJB Specification describes the extensions to the EJB Specification that arespecific to WebSphere Application Server. Use of these extensions issupported in VisualAge for Java only.

---

## Related information

For further information on the topics discussed in this manual, see thefollowing documents:

- Getting Started with WebSphere Application Server
- Building Business Solutions with WebSphere
- Component Broker Problem Determination Guide
- Component Broker System Administration Guide
- Component Broker Release Notes

---

## How to send your comments

Your feedback is important in helping to provide the most accurate andhighest quality information. If you have any comments about this book,send your comments by e-mail to wasdoc@us.ibm.com. Besure to include the name of the book, the document number of the book, theedition and version of WebSphere Application Server, and, if applicable, thespecific location of the information you are commenting on (for example, apage number or table number).

# An introduction to enterprise beans

This chapter looks at the characteristics and purpose of enterprisebeans. It describes the two basic types of enterprise beans and theirlife cycles, and it provides an example of how enterprise beans can becombined to create distributed, three-tiered applications.

---

## Bean basics

An enterprise bean is a Java component that can be combined with otherenterprise beans and other Java components to create a distributed,three-tiered application. There are two types of enterprisebeans:

- An *entity* bean encapsulates permanent data, which is stored ina data source such as a database or a file system, and associated methods tomanipulate that data. In most cases, an entity bean must be accessed insome transactional manner. Instances of an entity bean are unique andthey can be accessed by multiple users.

  For example, the information about a bank account can be encapsulated in anentity bean. An account entity bean might contain an account ID, anaccount type (checking or savings), and a balance variable and methods tomanipulate these variables.
- A *session* bean encapsulates ephemeral (nonpermanent) dataassociated with a particular EJB client. Unlike the data in an entitybean, the data in a session bean is not stored in a permanent data source, andno harm is caused if this data is lost. However, a session bean canupdate data in an underlying database, usually by accessing an entitybean. A session bean can also participate in a transaction.

  When created, instances of a session bean are identical, though somesession beans can store semipermanent data that makes them unique at certainpoints in their life cycle. A session bean is always associated with asingle client; attempts to make concurrent calls result in an exceptionbeing thrown.

  For example, the task associated with transferring funds between two bankaccounts can be encapsulated in a session bean. Such a transfer sessionbean can find two instances of an account entity bean (by using the accountIDs), and then subtract a specified amount from one account and add the sameamount to the other account.

### Entity beans

This section discusses the basics of entity beans.

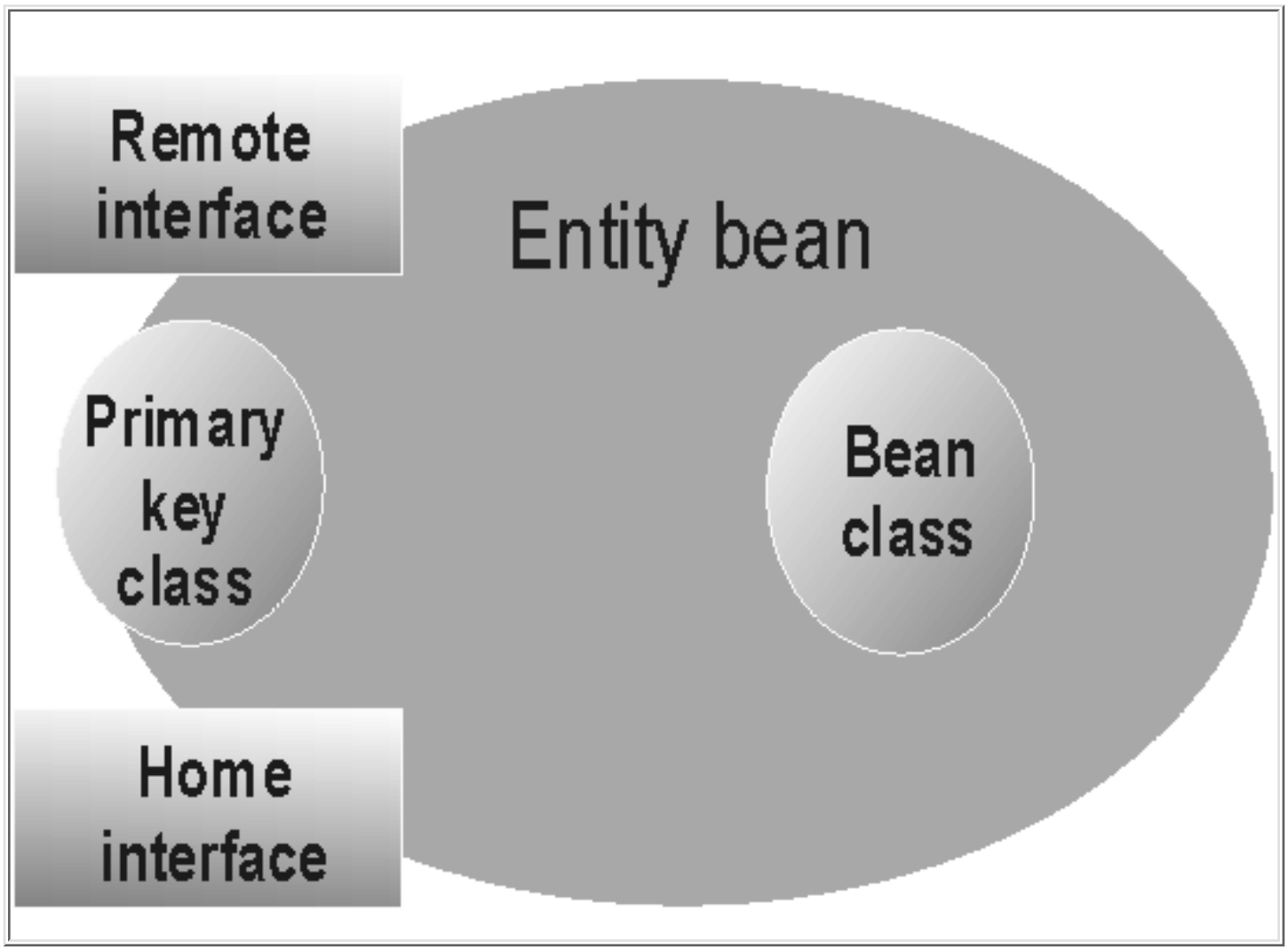
#### Basic components of an entity bean

Every entity bean must have the following components, which are illustrated inFigure 3:

- *Bean class*--This class encapsulates the data for theentity bean and contains the developer-implemented business methods thataccess the data. It also contains the methods used by the container tomanage the life cycle of an entity bean instance. EJB clients (whetherthey are other enterprise beans or user components such as servlets)*never* access objects of this class directly; instead, they usethe container-generated classes associated with the home and remote interfacesto manipulate the entity bean instance.
- *Home interface*--This interface defines the methods used bythe client to create, find, and remove instances of the entity bean.This interface is implemented by the container during deployment in a classknown generically as the *EJB home class*; instances arereferred to as *EJB home objects*.
- *Remote interface*--Once the client has used the homeinterface to gain access to an entity bean, it uses this interface to invokeindirectly the business methods implemented in the bean class. Thisinterface is implemented by the container during deployment in a class knowngenerically as the *EJB object class*; instances are referred toas *EJB objects*.

- *Primary key* -- One or more variables that uniquelyidentify a specific entity bean instance. A primary key that consistsof a single variable of a primitive Java data type can be specified atdeployment. A *primary key class* is used to encapsulateprimary keys that consist of multiple variables or more complex Java datatypes. The primary key class also contains methods to create primarykey objects and manipulate those objects.

**Figure 3. The components of an entity bean**




#### Data persistence

Entity beans encapsulate and manipulate *persistent* (or permanent)business data. For example, at a bank, entity beans can be used tomodel customer profiles, checking and savings accounts, car loans, mortgages,and customer transaction histories.

To ensure that this important data is not lost, the entity bean stores itsdata in a data source such as a database. When the data in anenterprise bean instance is changed, the data in the data source issynchronized with the bean data. Of course, this synchronization takesplace within the context of the appropriate type of transaction, so that if arouter goes down or a server fails, permanent changes are not lost.When you design an entity bean, you must decide whether you want theenterprise bean to handle this data synchronization or whether you want thecontainer to handle it. An enterprise bean that handles its own datasynchronization is said to implement *bean-managed persistence*(BMP), while an enterprise bean whose data synchronization is handled by thecontainer is said to implement *container-managed persistence*(CMP).

Unless you have a good reason for implementing BMP, it is recommended thatyou design your entity beans to use CMP. You must use entity beans withBMP if you want to use a data source that is not supported by the

EJBserver. The code for an enterprise bean with CMP is easier to write anddoes not depend on any particular data storage product, making it moreportable between EJB servers.

## Session beans

This section discusses the basics of session beans.

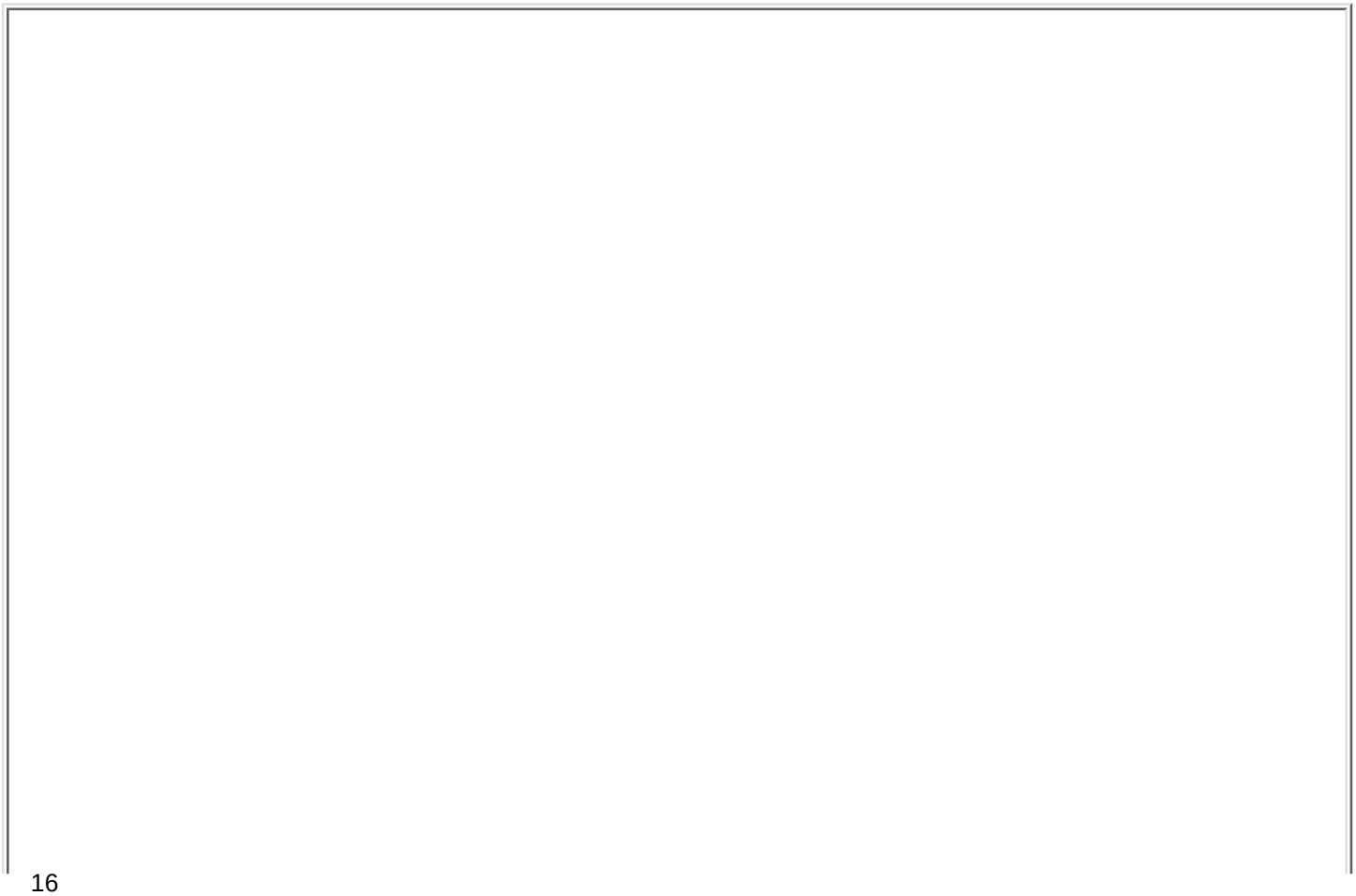
#### Basic components of a session bean

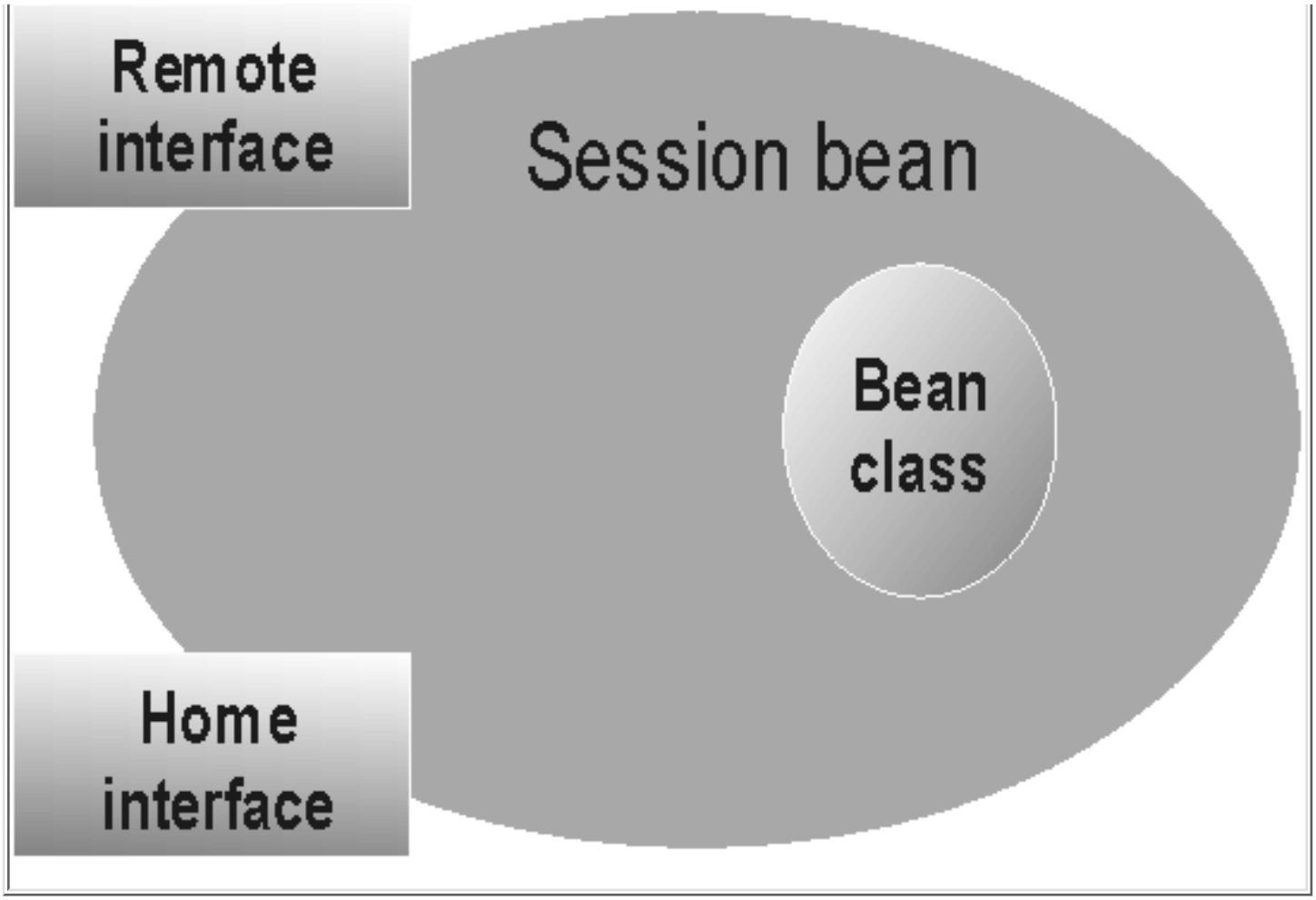
Every session bean must have the following components, which are illustratedin Figure 4:

- *Bean class*--This class encapsulates the data associatedwith the session bean and contains the developer-implemented business methodsthat access this data. It also contains the methods used by thecontainer to manage the life cycle of an session bean instance. EJBclients (whether they are other enterprise beans or user applications)*never* access objects of this class directly; instead, they usethe container-generated classes associated with the home and remote interfacesto manipulate the session bean.
- *Home interface*--This interface defines the methods used bythe client to create and remove instances of the session bean. Thisinterface is implemented by the container during deployment in a class knowngenerically as the *EJB home class*; instances are referred toas *EJB home object*.
- *Remote interface*--After the client has used the homeinterface to gain access to an session bean, it uses this interface to invokeindirectly the business methods implemented in the bean class. Thisinterface is implemented by the container during deployment in a class knowngenerically as the *EJB object class*; instances are referred toas *EJB objects*.

Unlike an entity bean, a session bean does not have a primary keyclass. A session bean does not require a primary key class because youdo not need to search for specific instances of session beans.

**Figure 4. The components of a session bean**






**Stateless versus stateful session beans**

Session beans encapsulate data and methods associated with a user session,task, or ephemeral object. By definition, the data in a session beaninstance is ephemeral; if it is lost, no real harm is done. Forexample, at a bank, a session bean represents a funds transfer, the creationof a customer profile or new account, and a withdrawal or deposit. Ifinformation about a fund transfer is already typed (but not yet committed),and a server fails, the balances of the bank accounts remains the same.Only the transfer data is lost, which can always be retyped.

The manner in which a session bean is designed determines whether its datais shorter lived or longer lived:

- If a session bean needs to maintain specific data across methods, it isreferred to as a *stateful* session bean. When a session beanmaintains data across methods, it is said to have a *conversationalstate*. A Web-based shopping cart is a classic use of a statefulsession bean. As the shopping cart user adds items to and subtractsitems from the shopping cart, the underlying session bean instance mustmaintain a record of the contents of the cart. After a particular EJBclient begins using an instance of a stateful session bean, the client mustcontinue to use that instance as long as the specific state of that instanceis required. If the session bean instance is lost before the contentsof the shopping cart are committed to an order, the shopper must load a newshopping cart.
- If a session bean does not need to maintain specific data across methods,it is referred to as a *stateless* session bean. The exampleTransfer session bean developed in Developing session beans provides an example of a stateless session bean. Forstateless session beans, a client can use any instance to invoke any of thesession bean's methods because all instances are the same.

---

## Creating an EJB module

The last step in the development of an enterprise bean is the creation of anEJB module. An EJB module consists of the following:

- One or more deployable enterprise beans.
- A deployment descriptor, stored in an Extensible Markup Language (XML)file. This file contains information about the structure and externaldependencies of the beans in the module, and application assembly informationdescribing how the beans are to be used in an application.

The EJB module can be created by using the tools within an integrateddevelopment environment (IDE) like IBM's VisualAge for Java EnterpriseEdition or by using the tools contained in WebSphere. For moreinformation, see Tools for developing and deploying enterprise beans in the EJB server (AE) environment.

For information about packaging enterprise beans for the EJB server (CB)environment, see Creating an EJB JAR file for an enterprise bean .

## The EJB module

The *EJB module* is used to assemble enterprise beans into a singledeployable unit; this file uses the standard Java archive fileformat. The EJB module can contain individual enterprise beans ormultiple enterprise beans. For more information, see Creating an EJB module and deployment descriptor.

## The deployment descriptor

The EJB module contains one or more deployable enterprise beans and one*deployment descriptor*. The deployment descriptor containsattribute and environment settings for each bean in the module, and it defineshow the container invokes functionality for all beans in the module.The deployment descriptor attributes can be set for the entire enterprise beanor for the individual methods in the bean. The container uses thedefinition of the bean-level attribute unless a method-level attribute isdefined, in which case the latter is used.The deployment descriptor contains the following information about entity andsession beans. These attributes can be set on the bean only; theycannot be set on a specific method of the bean.

- The bean's name, class, home interfaces, remote interfaces, and beantype (entity or session).
- *Primary key class* attribute--Identifies the primary keyclass for the bean. For more information, see Writing the primary key class (entity with CMP) or Writing or selecting the primary key class (entity with BMP).
- *Persistence management*. Specifies whether persistencemanagement is performed by the enterprise bean or by the container.
- *Container-managed fields* attribute--Lists those persistentvariables in the bean class that the container must synchronize with fields ina corresponding data source to ensure that this data is persistent andconsistent. For more information, see Defining variables.
- *Reentrant* attribute--Specifies whether an enterprise beancan invoke methods on itself or call another bean that invokes a method on thecalling bean. Only entity beans can be reentrant. For moreinformation, see Using threads and reentrancy in enterprise beans.
- *State management* attribute--Defines the conversationalstate of the session bean. This attribute must be set to eitherSTATEFUL or STATELESS. For more information on the meaning of theseconversational states, see Stateless versus stateful session beans.
- *Timeout* attribute--Defines the idle timeout value inseconds associated with this session bean. (This attribute is anextension to the standard deployment descriptor.)
- Settings for environment variables.
- References to external resources, such as resource factories, to the homesof other enterprise beans, and to

security roles.

The deployment descriptor contains the following application assemblyinformation:

- An application name and icons for identifying the module.
- The location of class files needed for a client program to access thebeans in the module.
- *Security roles*-- Define a group of permissions that agiven type of user must have in order to successfully use anapplication. Roles represent a type of user that has the same accessrights to an application.
- *Method permissions*--Define a permission to invoke aspecified group of methods of an enterprise bean's home and remoteinterfaces. This value is set per method.
- *Transaction* attributes--Define the transactional manner inwhich the container invokes a method for enterprise beans that requirecontainer-managed transaction demarcation. This value is set permethod. The values for this attribute are described in Enabling transactions and security in enterprise beans.
- *Transaction isolation level* attribute--Defines the degreeto which transactions are isolated from each other by the container.This value is set per method. The values for this attribute aredescribed in Enabling transactions and security in enterprise beans. (This attribute is an extension to the standarddeployment descriptor.)
- *RunAsMode* and *RunAsIdentity* attributes--The*RunAsMode* attribute defines the identity used to invoke themethod. If a specific identity is required, the*RunAsIdentity* attribute is used to specify that identity.This value is set per bean. The values for the *RunAsMode*attribute are described in Enabling transactions and security in enterprise beans. (This attribute is an extension to the standarddeployment descriptor.)

The following binding attribute is stored in the repository (it is not partof the deployment descriptor):

- *JNDI home name* attribute--Defines the Java Naming andDirectory Interface (JNDI) home name that is used to locate instances of anEJB home object. This value is set per bean. The values for thisrepository attribute are described in Creating and getting a reference to a bean's EJB object.

---

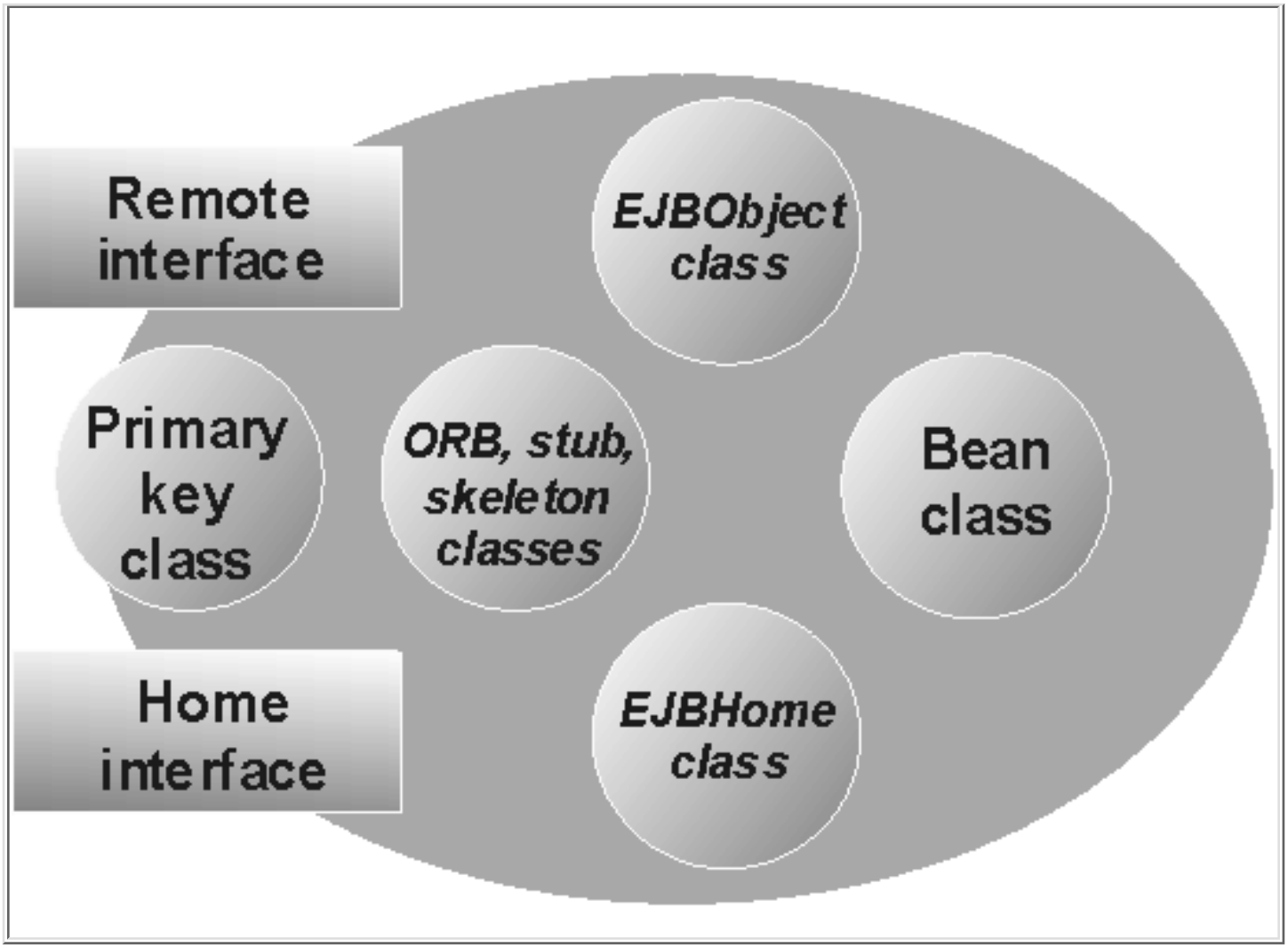
## Deploying an EJB module

When you deploy an EJB module, the deployment tool creates or incorporates thefollowing elements:

- The container-implemented *EJBObject* and *EJBHome*classes (hereafter referred to as the EJB object and EJB home classes) fromthe enterprise bean's home and remote interfaces (and the persistor andfinder classes for entity beans with CMP).
- The stub and skeleton files required for remote method invocation(RMI).

Figure 5 shows a simplified version of a deployed entity bean.

**Figure 5. The major components of a deployed entity bean**



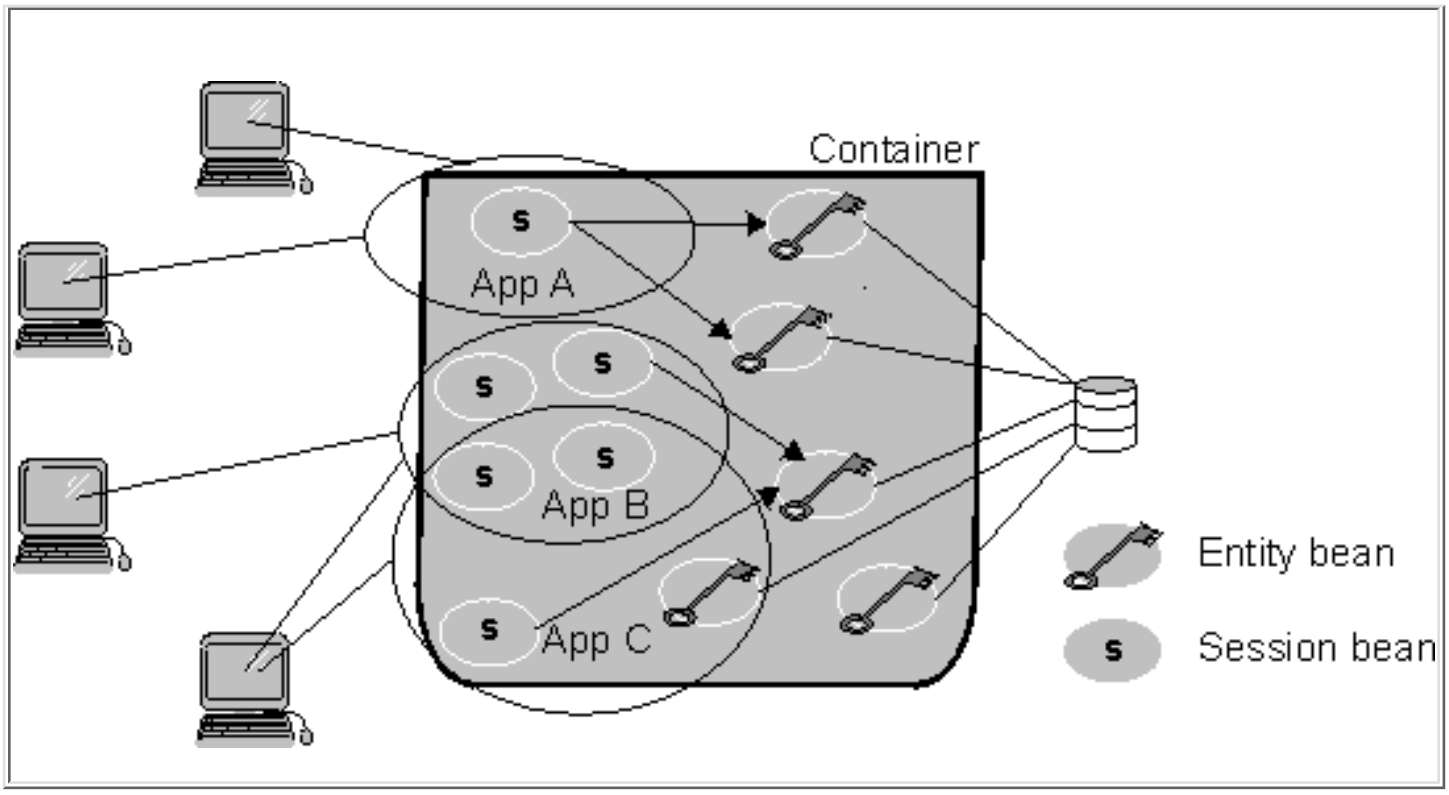

You can deploy an EJB module with a variety of different tools. Formore information, see Tools for developing and deploying enterprise beans in the EJB server (AE) environment or Tools for developing and deploying enterprise beans in the EJB server (CB) environment.

---

## Developing EJB applications

To create EJB applications, create the enterprise beans and EJB clients thatencapsulate your business data and functionality and then combine themappropriately. Figure 6 provides a conceptual illustration of how EJB applicationsare created by combining one or more session beans, one or more entity beans,or both. Although individual entity beans and session beans can be useddirectly in an EJB client, session beans are designed to be associated withclients and entity beans are designed to store persistent data, so most EJBapplications contain session beans that, in turn, access entity beans.

**Figure 6. Conceptual view of EJB applications**




This section provides an example of the ways in which enterprise beans canbe combined to create EJB applications.

## An example: enterprise beans for a bank

If you develop EJB applications for the banking industry, you can develop thefollowing entity beans to encapsulate your business data and associatedmethods:

- Account bean--An entity bean that contains information about customerchecking and savings accounts.
- CarLoan bean--An entity bean that contains information about anautomobile loan.
- Customer bean--An entity bean that contains information about acustomer, including information on accounts held and loans taken out by thecustomer.
- CustomerHistory bean--An entity bean that contains a record ofcustomer transactions for specified accounts.
- Mortgage bean--An entity bean that contains information about a homeor commercial mortgage.

An EJB client can directly access entity beans or session beans;however, the EJB Specification suggests that EJB clients use session beans toin turn access entity beans, especially in more complex applications.Therefore, as an EJB developer for the banking industry, you can create thefollowing session beans to represent client tasks:

- LoanApprover bean--A session bean that allows a loan to be approvedby using instances of the CarLoan bean, the Mortgage bean, or both.
- CarLoanCreator bean--A session bean that creates a new instance of aCarLoan bean.
- MortgageCreator bean--A session bean that creates a new instance of aMortgage bean.
- Deposit bean--A session bean that credits a specified amount to anexisting instance of an Account bean.
- StatementGenerator bean--A session bean that generates a statementsummarizing the activities associated with a customer's accounts by usingthe appropriate instances of the Customer and CustomerHistory entitybeans.
- Payment bean--A session bean that credits a payment to acustomer's loan by using instances of the

CarLoan bean, the Mortgagebean, or both.

- NewAccount bean--A session bean that creates a new instance of anAccount bean.
- NewCustomer bean--A session bean that creates a new instance of aCustomer bean.
- LoanReviewer bean--A session bean that accesses information about acustomer's outstanding loans (instances of the CarLoan bean, the Mortgagebean, or both).
- Transfer bean--A session bean that transfers a specified amountbetween two existing instances of an Account bean.
- Withdraw bean--A session bean that debits a specified amount from anexisting instance of an Account bean.

This example is simplified by necessity. Nevertheless, by using thisset of enterprise beans, you can create a variety of EJB applications fordifferent types of users by combining the appropriate beans within thatapplication. One or more EJB clients can then be built to access theapplication.

### Using the banking beans to develop EJB banking applications

When using beans built to the Sun Microsystems JavaBeans(TM) Specification(as opposed to the EJB Specification), you combine predefined components suchas buttons and text fields to create GUI applications. When usingenterprise beans, you combine predefined components such as the banking beansto create three-tiered applications.

For example, you can use the banking enterprise beans to create thefollowing EJB applications:

- Home Banking application--An Internet application that allows acustomer to transfer funds between accounts (with the Transfer bean), to makepayments on a loan by using funds in an existing account (with the Paymentbean), to apply for a car loan or home mortgage (with the CarLoanCreator beanor the MortgageCreator bean).
- Teller application--An intranet application that allows a teller tocreate new customer accounts (with the NewCustomer bean and the NewAccountbean), transfer funds between accounts (with the Transfer bean), and recordcustomer deposits and withdrawals (with the Withdraw bean and the Depositbean).
- Loan Officer application--An intranet application that allows a loanofficer to create and approve car loans and home mortgages (with theCarLoanCreator, MortgageCreator, LoanReviewer, and LoanApprover beans).
- Statement Generator application--A batch application that printsmonthly customer statements related to account activity (with theStatementGenerator bean).

These examples represent only a subset of the possible EJB applicationsthat can be created with the banking beans.

---

## Life cycles of enterprise bean instances

After an enterprise bean is deployed into a container, clients can create anduse instances of that bean as required. Within the container, instancesof an enterprise bean go through a defined life cycle. The events in anenterprise bean's life cycle are derived from actions initiated by eitherthe EJB client or the container in the EJB server. You must understandthis life cycle because for some enterprise beans, you must write some of thecode to handle the different events in the enterprise bean's lifecycle.

The methods mentioned in this section are discussed in greater detail in Developing enterprise beans.

### Session bean life cycle

This section describes the life cycle of a session bean instance.Differences between stateful and stateless session beans are noted.

#### Creation state

A session bean's life cycle begins when a client invokes a create methoddefined in the bean's home interface. In response to this methodinvocation, the container does the following:

1. Creates a new memory object for the session bean instance.
2. Invokes the session bean's setSessionContext method. (Thismethod passes the session bean instance a reference to a session contextinterface that can be used by the instance to obtain container services andget information about the caller of a client-invoked method.)
3. Invokes the session bean's ejbCreate method corresponding to thecreate method called by the EJB client.

#### Ready state

After a session bean instance is created, it moves to the ready state of itslife cycle. In this state, EJB clients can invoke the bean'sbusiness methods defined in the remote interface. The actions of thecontainer at this state are determined by whether a method is invokedtransactionally or nontransactionally:

- *Transactional method invocations*--When a client invokes atransactional business method, the session bean instance is associated with atransaction. After a bean instance is associated with a transaction, itremains associated until that transaction completes. (Furthermore, anerror results if an EJB client attempts to invoke another method on the samebean instance if invoking that method causes the container to associate thebean instance with another transaction or with no transaction.)

  The container then invokes the following methods:

  1. The afterBegin method, if that method is implemented by the beanclass.
  2. The business method in the bean class that corresponds to the businessmethod defined in the bean's remote interface and called by the EJBclient.
  3. The bean instance's beforeCompletion method, if that method isimplemented by the bean class and if a commit is requested prior to thecontainer's attempt to commit the transaction.

  The transaction service then attempts to commit the transaction, resultingeither in a commit or a roll back. When the transaction completes, thecontainer invokes the bean's afterCompletion method, passing thecompletion status of the transaction (either commit or rollback).

  If a rollback occurs, a stateful session bean can roll back itsconversational state to the values contained in the bean instance prior tobeginning the transaction. Stateless session beans do not maintain aconversational state, so they do not need to be concerned aboutrollbacks.
- *Nontransactional method invocations*--When a client invokesa nontransactional business method, the container simply invokes thecorresponding method in the bean class.

#### Pooled state

The container has a sophisticated algorithm for managing which enterprise beaninstances are retained in memory. When a container determines that astateful session bean instance is no longer required in memory, it invokes thebean instance's ejbPassivate method and moves the bean instance into areserve pool. A stateful session bean instance cannot be passivatedwhen it is associated with a transaction.

If a client invokes a method on a passivated instance of a stateful sessionbean, the container activates the instance by restoring the instance'sstate and then invoking the bean instance's ejbActivate method.When this method returns, the bean instance is again in the readystate.

Because every stateless session bean instance of a particular type is thesame as every other instance of that type,

stateless session bean instancesare not passivated or activated. These instances exist in a ready stateat all times until their removal.

#### Removal state

A session bean's life cycle ends when an EJB client or the containerinvokes a remove method defined in the bean's home interface and remoteinterface. In response to this method invocation, the container callsthe bean instance's ejbRemove method.

If you attempt to remove a bean instance while it is associated with atransaction, the javax.ejb.RemoveException is thrown.After a bean instance is removed, any attempt to invoke a method on thatinstance causes the java.rmi.NoSuchObjectException to bethrown.

A container can implicitly call a remove method on an instance after thelifetime of the EJB object has expired. The lifetime of a session EJBobject is set in the deployment descriptor with the *timeout*attribute.

For more information on the remove methods, see Removing a bean's EJB object.

## Entity bean life cycle

This section describes the life cycle of entity bean instances.Differences between entity beans with CMP and BMP are noted.

#### Creation State

An entity bean instance's life cycle begins when the container createsthat instance. After creating a new entity bean instance, the containerinvokes the instance's setEntityContext method. This method passesthe bean instance a reference to an entity context interface that can be usedby the instance to obtain container services and get information about thecaller of a client-invoked method.

#### Pooled State

After an entity bean instance is created, it is placed in a pool of availableinstances of the specified entity bean class. While the instance is inthis pool, it is not associated with a specific EJB object. Everyinstance of the same enterprise bean class in this pool is identical.While an instance is in this pooled state, the container can use it to invokeany of the bean's finder methods.

#### Ready State

When a client needs to work with a specific entity bean instance, thecontainer picks an instance from the pool and associates it with the EJBobject initialized by the client. An entity bean instance is moved fromthe pooled to the ready state if there are no available instances in the readystate.

There are two events that cause an entity bean instance to be moved fromthe pooled state to the ready state:

- When a client invokes the create method in the bean's home interfaceto create a new and unique entity of the entity bean class (and a new recordin the data source). As a result of this method invocation, thecontainer calls the bean instance's ejbCreate and ejbPostCreate methods,and the new EJB object is associated with the bean instance.
- When a client invokes a finder method to manipulate an existing instanceof the entity bean class (associated with an existing record in the datasource). In this case, the container calls the bean instance'sejbActivate method to associate the bean instance with the existing EJBobject.

When an entity bean instance is in the ready state, the container caninvoke the instance's ejbLoad and ejbStore methods to synchronize thedata in the instance with the corresponding data in the data source. Inaddition, the client can invoke the bean instance's business methods whenthe instance is in this state. All interactions required

to handle anentity bean instance's business methods in the appropriate transactional(or nontransactional) manner are handled by the container.

When a container determines that an entity bean instance in the ready stateis no longer required, it moves the instance to the pooled state. Thistransition to the pooled state results from either of the followingevents:

- When the container invokes the ejbPassivate method.
- When the EJB client invokes a remove method on the EJB object or on theEJB home object. When one of these methods is called, the underlyingentity is removed permanently from the data source.

#### Removal State

An entity bean instance's life cycle ends when the container invokes theunsetEntityContext method on an entity bean instance in the pooledstate. Do not confuse the removal of an entity bean instance with theremoval of the underlying entity whose data is stored in the datasource. The former simply removes an uninitialized object; thelatter removes data from the data source.

For more information on the remove methods, see Removing a bean's EJB object.

# WebSphere Programming Model Extensions

This section discusses facilities that are provided as part of theProgramming Model Extensions in WebSphere Application Server:

- The exception-chaining package, which can be used by distributedapplications to capture a sequence of exceptions. For more information,see The distributed-exception package.
- The command package, which can be used by distributed applications toreduce the number of remote invocations they must make. For moreinformation, see The command package.
- The localizable-text package, which can be used by distributedapplications spanning locales to deliver output in a user-specifiedlanguage. For more information, see The localizable-text package.

The exception-chaining and command packages are available as part ofWebSphere Application Server Advanced Edition and Enterprise Edition; thelocalizable-text package is available as part of WebSphere Application ServerAdvanced Edition. All three packages are general-purpose utilities,designed to provide common functions in a reusable way. Although thesefacilities are described in the context of enterprise beans, they areavailable to any WebSphere Application Server Java application. Theyare not restricted to use with enterprise beans.

---

## The distributed-exception package

Distributed applications require a strategy for exception handling.As applications become more complex and are used by more participants,handling exceptions becomes problematic. To capture the informationcontained in every exception, methods have to rethrow every exception theycatch. If every method adopts this approach, the number of exceptionscan become unmanageable, and the code itself becomes less maintainable.Furthermore, if a new method introduces a new exception, all existing methodsthat call the new method have to be modified to handle the newexception. Trying to explicitly manage every possible exception in acomplex application quickly becomes intractable.

In order to keep the number of exceptions manageable, some programmersadopt a strategy in which methods catch all exceptions in a single clause andthrow one exception in response. This reduces the number of exceptionseach method must recognize, but it also means that the information about theoriginating exception is lost. This loss of information can bedesirable, for example, when you wish to hide implementation details from endusers. However, this strategy can make applications much more difficultto debug.

The distributed-exception package provides a facility that allows you tobuild chains of exceptions. An *exception chain* encapsulatesthe stack of previous exceptions. With an exception chain, you canthrow one exception in response to another without discarding the previousexceptions, so you can manage the number of exceptions without losing theinformation they carry. Exceptions that support chaining are called*distributed exceptions*.

### Overview

Support for chaining distributed exceptions is provided by thecom.ibm.websphere.exception Java package. Thefollowing classes and interfaces make up this package:

- DistributedException--This class provides access to the methods onthe DistributedExceptionInfo object. It acts as the root class forexceptions in a distributed application. For more information, see The DistributedException class.
- DistributedExceptionEnabled--This interface allows exceptions thatcannot inherit from the DistributedException class to be used in exceptionchains, so that exceptions based on predefined exceptions can becaptured. For more information, see The DistributedExceptionEnabled interface.
- DistributedExceptionInfo--This class encapsulates the work necessaryfor distributed exceptions. An exception class that extends theDistributedException class automatically gets access to this class. Aclass that implements the DistributedExceptionEnabled interface mustexplicitly declare a DistributedExceptionInfo attribute. For moreinformation, see The DistributedExceptionInfo class.
- ExceptionInstantiationException--This class defines the exceptionthat is thrown if an exception chain cannot be created. This exceptionis instantiated internally, but you can catch and re-throw it.

This section provides a general description of the interfaces and classesin the exception-chaining package.

#### The DistributedException class

The DistributedException class provides the root exception for exceptionhierarchies defined by applications. With this class, you build chainsof exceptions by saving a caught exception and bundling it into the newexception to be thrown. This way, the information about the oldexception is forwarded along with the new exception. The class declaressix constructors; Figure 71 shows the signatures for these constructors. Whenyour exception is a subclass of the DistributedException class, you mustprovide corresponding constructors in your exception class.

**Figure 71. Code example: Constructors for the DistributedException class**

```
...public class DistributedException extends Exceptionimplements DistributedExceptionEnabled{    //
Constructors    public DistributedException() {...}    public DistributedException(String message)
{...}    public DistributedException(Throwable exception) {...}    public
DistributedException(String message,Throwable exception) {...}    public DistributedException(String
resourceBundleName,                                String resourceKey,
Object[] formatArguments,                                String defaultText)     {...}    public
DistributedException(String resourceBundleName,                                String resourceKey,
Object[] formatArguments,                                String defaultText,
Throwable exception)     {...}    // Other methods    ...}
```

The class also provides methods for extracting exceptions from the chain andquerying the chain. These methods include:

- getMessage--This method returns the message string associated withthe current exception.
- getPreviousException--This method returns the preceding exception ina chain as a Throwable object. If there are no previous exceptions, itreturns null.
- getOriginalException--This method returns the original exception in achain as a Throwable object. If there is no prior exception, it returnsnull.
- getException--This method returns the most recent instance of thenamed exception from the chain as a Throwable object. If there are noinstances present, it returns null.
- getExceptionInfo--This method returns the DistributedExceptionInfoobject for the exception.
- printStackTrace--These methods print the stack trace for the currentexception, which includes the stack traces of all previous exceptions in thechain.

Localization supportSupport for localized messages is provided by two of the constructors fordistributed exceptions. These constructors take arguments representinga resource bundle, a resource key, a default message, and the set ofreplacement strings for variables in the message. A resource bundle isa collection of resources or resource names representing informationassociated with a specific locale. Resource bundles are provided aseither a subclass of the ResourceBundle class or in a properties file.The resource key indicates which resource in the bundle to retrieve.The default message is returned if either the name of the resource bundle orthe key is null or invalid.

#### The DistributedExceptionEnabled interface

Use the DistributedExceptionEnabled interface to create distributed exceptionswhen your exception cannot extend the DistributedException class.Because Java does not permit multiple inheritance, you cannot extend multipleexception classes. If you are extending an existing exception class,for example, javax.ejb.CreateException, you cannot also extendthe DistributedException class. To allow your new exception class tochain other exceptions, you must implement the DistributedExceptionEnabledinterface instead.The DistributedExceptionEnabled interface declares eight methods you mustimplement in your exception class:

- getMessage--This method returns the message string associated withthe current exception.
- getPreviousException--This method returns the preceding exception ina chain as a Throwable object. If there are no previous exceptions, itreturns null.
- getOriginalException--This method returns the original exception in achain as a Throwable object. If there is no prior exception, it returnsnull.
- getException--This method returns the most recent instance of thenamed exception from the chain as a Throwable object. If there are noinstances present, it returns null.
- getExceptionInfo--This method returns the DistributedExceptionInfoobject for the exception.
- printStackTrace--These methods print the stack trace for the currentexception, which includes the stack traces of all previous exceptions in thechain.
- printSuperStackTrace--This method is used by aDistributedExceptionInfo object to retrieve and save the current stacktrace.

When implementing the DistributedExceptionEnabled interface, you mustdeclare a DistributedExceptionInfo attribute. This attribute providesimplementations for most of these methods, so implementing them in yourexception class consists of calling the corresponding methods on theDistributedExceptionInfo object. For more information, see Implementing the methods from the DistributedExceptionEnabled interface.

#### The DistributedExceptionInfo class

The DistributedExceptionInfo class provides the functionality required fordistributed exceptions. It must be used by any exception thatimplements the DistributedExceptionEnabled interface (which includes theDistributedException class). A DistributedExceptionInfo object containsthe exception itself, and it provides constructors for creating exceptionchains and methods for retrieving the information within those chains.It also provides the underlying methods for managing chainedexceptions.

## Extending the DistributedException class

The DistributedException class provides the root exception for exceptionhierarchies defined by applications. The class also provides methodsfor extracting exceptions from the chain and querying the chain. Youmust provide constructors corresponding to the constructors in theDistributedException class (see Figure 71). The constructors can simply pass arguments to theconstructor in the DistributedException class by using super methods, asillustrated in Figure 72.

**Figure 72. Code example: Constructors in an exception class that extends the DistributedException class**

```
...import com.ibm.websphere.exception.*;public class MyDistributedException extends
DistributedException{    // Constructors    public MyDistributedException() {        super();    }
public MyDistributedException(String message) {        super(message);    }    public
MyDistributedException(Throwable exception) {        super(exception);    }    public
MyDistributedException(String message, Throwable exception) {        super(message, exception);    }
public MyDistributedException(String resourceBundleName,                    String resourceKey,
Object[] formatArguments,                    String defaultText)     {
super(resourceBundleName, resourceKey, formatArguments, defaultText);    }    public
MyDistributedException(String resourceBundleName,                    String resourceKey, Object[]
formatArguments,                    String defaultText, Throwable exception)    {
super(resourceBundleName, resourceKey, formatArguments, defaultText,            exception);    }}
```

## Implementing the DistributedExceptionEnabled interface

Use the DistributedExceptionEnabled interface to create distributed exceptionswhen your exception cannot extend the DistributedException class. Toallow your new exception class to be chained, you must implement theDistributedExceptionEnabled interface instead. Figure 73 shows the structure of an exception class that extends theexisting javax.ejb.CreateException class and implements theDistributedExceptionEnabled interface. The class also declares therequired DistributedExceptionInfo object.

**Figure 73. Code example: The structure of an exception class that implements the DistributedExceptionEnabled interface**

```
...import javax.ejb.*;import com.ibm.websphere.exception.*;public class AccountCreateException
extends CreateExceptionimplements DistributedExceptionEnabled{    DistributedExceptionInfo
exceptionInfo = null;    // Constructors    ...    // Methods from the DistributedExceptionEnabled
interface    ...}
```

#### Implementing the constructors for the exception class

The exception-chaining package supports six different ways of creatinginstances of exception classes (see Figure 71). When you write an exception class by implementingthe DistributedExceptionEnabled interface, you must implement theseconstructors. In each one, you must use the DistributedExceptionInfoobject to capture the information for chaining the exception. Figure 74 shows standard implementations for the sixconstructors.

**Figure 74. Code example: Constructors for an exception class that implements the DistributedExceptionEnabled interface**

```
...public class AccountCreateException extends CreateExceptionimplements
DistributedExceptionEnabled{    DistributedExceptionInfo exceptionInfo = null;    // Constructors
AccountCreateException() {        super ();        exceptionInfo = new
DistributedExceptionInfo(this);    }    AccountCreateException(String msg) {        super (msg);
exceptionInfo = new DistributedExceptionInfo(this);    }    AccountCreateException(Throwable e) {
super ();        exceptionInfo = new DistributedExceptionInfo(this, e);    }
AccountCreateException(String msg, Throwable e) {        super (msg);        exceptionInfo = new
DistributedExceptionInfo(this, e);    }    AccountCreateException(String resourceBundleName, String
resourceKey,                                Object[] formatArguments, String defaultText)    {
super ();        exceptionInfo = new DistributedExceptionInfo(resourceBundleName,
resourceKey, formatArguments, defaultText, this);    }   AccountCreateException(String
resourceBundleName, String resourceKey,                                    Object[] formatArguments,
String defaultText,                              Throwable exception)    {        super ();
exceptionInfo = new DistributedExceptionInfo(resourceBundleName,                      resourceKey,
formatArguments, defaultText, this, exception);    }    // Methods from the
DistributedExceptionEnabled interface    ...}
```

#### Implementing the methods from the DistributedExceptionEnabled interface

The DistributedExceptionInfo object provides implementations for most of themethods in the DistributedExceptionEnabled interface, so you can implement therequired methods in your exception class by calling the corresponding methodson the DistributedExceptionInfo object. Figure 75 illustrates this technique. The only two methods thatdo not involve calling a corresponding method on the DistributedExceptionInfoobject are the getExceptionInfo method, which returns the object, and theprintSuperStackTrace method, which calls the super.printStackTracemethod.

**Figure 75. Code example: Implementations of the methods in the DistributedExceptionEnabled interface**

```
...public class AccountCreateException extends CreateExceptionimplements
DistributedExceptionEnabled{    DistributedExceptionInfo exceptionInfo = null;    // Constructors
...    // Methods from the DistributedExceptionEnabled interface    String getMessage() {        if
(exceptionInfo != null)            return exceptionInfo.getMessage();        else return null;    }
Throwable getPreviousException() {        if (exceptionInfo != null)            return
exceptionInfo.getPreviousException();        else return null;    }    Throwable
getOriginalException() {        if (exceptionInfo != null)            return
exceptionInfo.getOriginalException();        else return null;    }    Throwable getException(String
exceptionClassName) {        if (exceptionInfo != null)            return
exceptionInfo.getException(exceptionClassName);        else return null;    }
DistributedExceptionInfo getExceptionInfo() {        if (exceptionInfo != null)            return
exceptionInfo;        else return null;    }    void printStackTrace() {        if (exceptionInfo !=
null)            return exceptionInfo.printStackTrace();        else return null;    }    void
printStackTrace(PrintWriter pw) {        if (exceptionInfo != null)            return
exceptionInfo.printStackTrace(pw);        else return null;    }    void
printSuperStackTrace(PrintWriter pw)        if (exceptionInfo != null)            return
super.printStackTrace(pw);        else return null;    }}
```

## Using distributed exceptions

Defining a distributed exception gives you the ability to chain exceptionstogether. The DistributedExceptionInfo class provides methods foradding information to an exception chain and for extracting information fromthe chain. This section illustrates the use of distributedexceptions.

#### Catching distributed exceptions

You can catch exceptions that extend the DistributedException class orimplement the DistributedExceptionEnabled interface separately. You canalso test a caught exception to see if it has implemented theDistributedExceptionEnabled interface. If it has, you can treat it asany other distributed exception. Figure 76 shows the use of the instanceof method to test for exceptionchaining.

**Figure 76. Code example: Testing for an exception that implements the DistributedExceptionEnabled interface**

```
....try {    someMethod();}catch (Exception e) {    ...    if (e instanceof
DistributedExceptionEnabled) {            ...    }...
```

#### Adding an exception to a chain

To add an exception to a chain, you must call one of the constructors foryour distributed-exception class. This captures the previous exceptioninformation and packages it with the new exception. Figure 77 shows the use of the MyDistributedException(Throwable)constructor.

**Figure 77. Code example: Adding an exception to a chain**

```
void someMethod() throws MyDistributedException {    try {        someOtherMethod();    }    catch
(DistributedExceptionEnabled e) {        throw new MyDistributedException(e);    }    ...}...
```

#### Retrieving information from a chain

Chained exceptions allow you to retrieve information about prior exceptionsin the chain. For example, the getPreviousException,getOriginalException, and getException(String) methods allow you to retrievespecific exceptions from the chain. You can retrieve the messageassociated with the current exception by calling the getMessage method.You can also get information about the entire chain by calling one of theprintStackTrace methods. Figure 78 illustrates calling the getPreviousException andgetOriginalException methods.

**Figure 78. Code example: Extracting exceptions from a chain**

```
...try {    someMethod();}catch (DistributedExceptionEnabled e) {    try {       Throwable prev =
e.getPreviousException();    }    catch (ExceptionInstantiationException eie) {
DistributedExceptionInfo prevExInfo = e.getPreviousExceptionInfo();        if (prevExInfo != null) {
String prevExName = prevExInfo.getClassName();            String prevExMsg =
prevExInfo.getClassMessage();            ...        }    }    try {        Throwable orig =
e.getOriginalException();    }    catch (ExceptionInstantiationException eie) {
DistributedExceptionInfo origExInfo = null;        DistributedExceptionInfo prevExInfo =
e.getPreviousExceptionInfo();        while (prevExInfo != null) {            origExInfo =
prevExInfo;            prevExInfo = prevExInfo.getPreviousExceptionInfo();        }        if
(origExInfo != null) {            String origExName = origExInfo.getClassName();            String
origExMsg = origExInfo.getClassMessage();        ...        }    }}...
```

---

## The command package

Distributed applications are defined by the ability to utilize remoteresources as if they were local, but this remote work affects the performanceof distributed applications. Distributed applications can improveperformance by using remote calls sparingly. For example, if a serverdoes several tasks for a client, the application can run more quickly if theclient bundles requests together, reducing the number of individual remotecalls. The command package provides a mechanism for collecting sets ofrequests to be submitted as a unit.

In addition to giving you a way to reduce the number of remote invocationsa client makes, the command package provides a generic way of makingrequests. A client instantiates the command, sets its input data, andtells it to run. The command infrastructure determines the targetserver and passes a copy of the command to it. The server runs thecommand, sets any output data, and copies it back to the client. Thepackage provides a common way to issue a command, locally or remotely, andindependently of the server's implementation. Any server (anenterprise bean, a Java Database Connectivity (JDBC) server, a servlet, and soon) can be a target of a command if the server supports Java access to itsresources and provides a way to copy the command between the client'sJava Virtual Machine (JVM) and its own JVM.

### Overview

The command facility is implemented in thecom.ibm.websphere.command Java package. Theclasses and interfaces in the command package fall into four generalcategories:

- Interfaces for creating commands. For more information, see Facilities for creating commands.
- Classes and interfaces for implementing commands. For moreinformation, see Facilities for implementing commands.

- Classes and interfaces for determining where the command is run.For more information, see Facilities for setting and determining targets.
- Classes defining package-specific exceptions. For more information,see Exceptions in the command package.

This section provides a general description of the interfaces and classesin the command package.

#### Facilities for creating commands

The Command interface specifies the most basic aspects of a command.This interface is extended by both the TargetableCommand interface and theCompensableCommand interface, which offer additional features. Tocreate commands for applications, you must:

- Define an interface that extends one or more of interfaces in the commandpackage.
- Provide an implementation class for your interface.

In practice, most commands implement the TargetableCommand interface, whichallows the command to be executed remotely. Figure 79 shows the structure of a command interface for a targetablecommand.

**Figure 79. Code example: The structure of an interface for a targetable command**

```
...import com.ibm.websphere.command.*;public interface MySimpleCommand extends TargetableCommand {
// Declare application methods here}
```

The CompensableCommand interface allows the association of one command withanother that can undo the work of the first. Compensable commands alsotypically implement the TargetableCommand interface. Figure 80 shows the structure of a command interface for a targetable,compensable command.

**Figure 80. Code example: The structure of an interface for a targetable, compensable command**

```
...import com.ibm.websphere.command.*;public interface MyCommand extends TargetableCommand,
CompensableCommand {       // Declare application methods here}
```

#### Facilities for implementing commands

Commands are implemented by extending the class TargetableCommandImpl, whichimplements the TargetableCommand interface. The TargetableCommandImplclass is an abstract class that provides some implementations for some of themethods in the TargetableCommand interface (for example, setting returnvalues) and declares additional methods that the application itself mustimplement (for example, how to execute the command).

You implement your command interface by writing a class that extends theTargetableCommandImpl class and implements your command interface. Thisclass contains the code for the methods in your interface, the methodsinherited from extended interfaces (the TargetableCommand andCompensableCommand interfaces), and the required (abstract) methods in theTargetableCommandImpl class. You can also override the defaultimplementations of other methods provided in the TargetableCommandImplclass. Figure 81 shows the structure of an implementation class for theinterface in Figure 80.

**Figure 81. Code example: The structure of an implementation class for a command interface**

```
...import java.lang.reflect.*;import com.ibm.websphere.command.*;public class MyCommandImpl extends
TargetableCommandImplimplements MyCommand {      // Set instance variables here     ...     //
Implement methods in the MyCommand interface     ...     // Implement methods in the
CompensableCommand interface     ...     // Implement abstract methods in the TargetableCommandImpl
class     ...}
```

#### Facilities for setting and determining targets

The object that is the target of a TargetableCommand must implement theCommandTarget interface. This object can be an actual server-sideobject, like an entity bean, or it can be a client-side adapter for aserver. The implementor of the CommandTarget interface is responsiblefor ensuring the proper execution of a command in the desired target serverenvironment. This typically requires the following steps:

1. Copying the command to the target server by using a server-specificprotocol.
2. Running the command in the server.
3. Copying the executed command from the target server to the client by usinga server-specific protocol.

Common ways to implement the CommandTarget interface include:

- A local target, which runs in the client's JVM.
- A client-side adapter for a server. For an example that implementsthe target as a client-side adapter, see Writing a command target (client-side adapter).
- An enterprise bean (either a session bean or an entity bean). Figure 82 shows the structure of the remote interface and enterprisebean class for an entity bean that implements the CommandTargetinterface.

**Figure 82. Code example: The structure of a command-target entity bean**

```
...import java.rmi.RemoteException;import java.util.Properties;import javax.ejb.*;import
com.ibm.websphere.command.*;// Remote interface for the MyBean enterprise bean (also a command
target)public interface MyBean extends EJBObject, CommandTarget {      // Declare methods for the
remote interface     ...}// Entity bean class for the MyBean enterprise bean (also a command
target)public class MyBeanClass implements EntityBean, CommandTarget {      // Set instance
variables here     ...     // Implement methods in the remote interface     ...     // Implement
methods in the EntityBean interface     ...     // Implement the method in the CommandTarget
interface     ...}
```

Since targetable commands can be run remotely in another JVM, the commandpackage provides mechanisms for determining where to run the command. A*target policy* associates a command with a target and is specifiedthrough the TargetPolicy interface. You can design customized targetpolicies by implementing this interface, or you can use the providedTargetPolicyDefault class. For more information, see Targets and target policies.

#### Exceptions in the command package

The command package defines a set of exception classes. TheCommandException class extends the DistributedException class and acts as thebase class for the additional command-related exceptions:UnauthorizedAccessException, UnsetInputPropertiesException, andUnavailableCompensableCommandException. Applications can extend theCommandException class to define additional exceptions, as well.

Although the CommandException class extends the DistributedException class,you do not have to import the distributed-exception package,com.ibm.websphere.exception, unless you need to use thefeatures of the DistributedException class in your application. Formore information on distributed exceptions, see The distributed-exception package.

## Writing command interfaces

To write a command interface, you extend one or more of the three interfacesincluded in the command package. The base interface for all commands isthe Command interface. This interface provides only the client-sideinterface for generic commands and declares three basic methods:

- isReadyToCallExecute--This method is called on the client side beforethe command is passed to the server for execution.
- execute--This method passes the command to the target and returns anydata.
- reset--This method reverts any output properties to the values theyhad before the execute method was called so that the object can bereused.

The implementation class for your interface must contain implementationsfor the isReadyToCallExecute and reset methods. The execute method isimplemented for you elsewhere; for more information, see Implementing command interfaces. Most commands do not extend the Command interfacedirectly but use one of the provided extensions: the TargetableCommandinterface and the CompensableCommand interface.

#### The TargetableCommand interface

The TargetableCommand interface extends the Command interface and provides forremote execution of commands. Most commands will be targetablecommands. The TargetableCommand interface declares several additionalmethods:

- setCommandTarget--This method allows you to specify the target objectto a command.
- setCommandTargetName--This method allows you to specify the target byname to a command.
- getCommandTarget--This method returns the target object of thecommand.
- getCommandTargetName--This method returns the name of the targetobject of the command.
- hasOutputProperties--This method indicates whether or not the commandhas output that must be copied back to the client. (The implementationclass also provides a method, setHasOutputProperties, for setting the outputof this method. By default, hasOutputProperties returns true.)
- setOutputProperties--This method saves output values from the commandfor return to the client.
- performExecute--This method encapsulates the application-specificwork. It is called for you by the execute method declared in theCommand interface.

With the exception of the performExecute method, which you must implement,all of these methods are implemented in the TargetableCommandImplclass. This class also implements the execute method declared in theCommand interface.

#### The CompensableCommand interface

The CompensableCommand interface also extends the Command interface. Acompensable command is one that has another command (a compensator) associatedwith it, so that the work of the first can be undone by thecompensator. For example, a command that attempts to make an airlinereservation followed by a hotel reservation can offer a compensating commandthat allows the user to cancel the airline reservation if the hotelreservation cannot be made.

The CompensableCommand interface declares one method:

- getCompensatingCommand--This methods returns the command that can beused to undo the effects of the original command.

To create a compensable command, you write an interface that extends theCompensableCommand interface. Such interfaces typically extend theTargetableCommand interface as well. You must implement thegetCompensatingCommand method in the implementation class for your interface.

You must also implement the compensating command.

#### The example application

The example used throughout the remainder of this discussion uses an entitybean with container-managed persistence (CMP) called CheckingAccountBean,which allows a client to deposit money, withdraw money, set a balance, get abalance, and retrieve the name on the account. This entity bean alsoaccepts commands from the client. The code examples illustrate thecommand-related programming. For a servlet-based example, see Writing a command target (client-side adapter).

Figure 83 shows the interface for the ModifyCheckingAccountCmdcommand. This command is both targetable and compensable, so theinterface extends both TargetableCommand and CompensableCommandinterfaces.

**Figure 83. Code example: The ModifyCheckingAccountCmd interface**

```
...import com.ibm.websphere.exception.*;import com.ibm.websphere.command.*;public interface
ModifyCheckingAccountCmdextends TargetableCommand, CompensableCommand {       float getAmount();
float getBalance();      float getOldBalance();       // Used for compensating      float
setBalance(float amount);      float setBalance(int amount);      CheckingAccount
getCheckingAccount();      void setCheckingAccount(CheckingAccount newCheckingAccount);
TargetPolicy getCmdTargetPolicy();      ...}
```

## Implementing command interfaces

The command package provides a class, TargetableCommandImpl, that implementsall of the methods in the TargetableCommand interface except theperformExecute method. It also implements the execute method from theCommand interface. To implement an application's commandinterface, you must write a class that extends the TargetableCommandImpl classand implements your command interface. Figure 84 shows the structure of the ModifyCheckingAccountCmdImplclass.

**Figure 84. Code example: The structure of the ModifyCheckingAccountCmdImpl class**

```
...public class ModifyCheckingAccountCmdImpl extends TargetableCommandImplimplements
ModifyCheckingAccountCmd{    // Variables    ...    // Methods    ...}
```

The class must declare any variables and implement these methods:

- Any methods you defined in your command interface.
- The isReadyToCallExecute and reset methods from the Commandinterface.
- The performExecute method from the TargetableCommand interface.
- The getCompensatingCommand method from the CompensableCommand interface,if your command is compensable. You must also implement thecompensating command.

You can also override the nonfinal implementations provided in theTargetableCommandImpl class. The most likely candidate forreimplementation is the setOutputProperties method, since the defaultimplementation does not save final, transient, or static fields.

#### Defining instance and class variables

The ModifyCheckingAccountCmdImpl class declares the variables used by themethods in the class, including the remote interface of the CheckingAccountentity bean; the variables used to capture operations on the checkingaccount (balances and amounts); and a compensating command. Figure 85 shows the variables used by the ModifyCheckingAccountCmdcommand.

**Figure 85. Code example: The variables in the ModifyCheckingAccountCmdImpl class**

```
...public class ModifyCheckingAccountCmdImpl extends TargetableCommandImplimplements
ModifyCheckingAccountCmd{    // Variables    public float balance;    public float amount;    public
float oldBalance;    public CheckingAccount checkingAccount;    public
ModifyCheckingAccountCompensatorCmd                    modifyCheckingAccountCompensatorCmd;
...}
```

#### Implementing command-specific methods

The ModifyCheckingAccountCmd interface defines several command-specificmethods in addition to extending other interfaces in the commandpackage. These command-specific methods are implemented in theModifyCheckingAccountCmdImpl class.

You must provide a way to instantiate the command. The commandpackage does not specify the mechanism, so you can choose the technique mostappropriate for your application. The fastest and most efficienttechnique is to use constructors. The most flexible technique is to usea factory. Also, since commands are implemented internally as JavaBeanscomponents, you can use the standard Beans.instantiate method.The ModifyCheckingAccountCmd command uses constructors.

Figure 86 shows the two constructors for the command. Thedifference between them is that the first uses the default target policy fordetermining the

target of the command and the second allows you to specify acustom policy. (For more information on targets and target policies,see Targets and target policies.)

Both constructors take a CommandTarget object as an argument and cast it tothe CheckingAccount type. The CheckingAccount interface extends boththe CommandTarget interface and the EJBObject (see Figure 95). The resulting checkingAccount object routes thecommand to the desired server by using the bean's remoteinterface. (For more information on CommandTarget objects, see Writing a command target (server).)

**Figure 86. Code example: Constructors in the ModifyCheckingAccountCmdImpl class**

```
...public class ModifyCheckingAccountCmdImpl extends TargetableCommandImplimplements
ModifyCheckingAccountCmd{    // Variables    ...    // Constructors    // First constructor: relies
on the default target policy    public ModifyCheckingAccountCmdImpl(CommandTarget target,
float newAmount)    {        amount = newAmount;        checkingAccount = (CheckingAccount)target;
setCommandTarget(target);    }    // Second constructor: allows you to specify a custom target
policy    public ModifyCheckingAccountCmdImpl(CommandTarget target,                    float
newAmount,                    TargetPolicy targetPolicy)    {        setTargetPolicy(targetPolicy);
amount = newAmount;        checkingAccount = (CheckingAccount)target;
setCommandTarget(target);     }    ...}
```

Figure 87 shows the implementation of the command-specificmethods:

- setBalance--This method sets the balance of the account.
- getAmount--This method returns the amount of a deposit orwithdrawal.
- getOldBalance, getBalance--These methods capture the balance beforeand after an operation.
- getCmdTargetPolicy--This method retrieves the current targetpolicy.
- setCheckingAccount, getCheckingAccount--These methods set andretrieve the current checking account.

**Figure 87. Code example: Command-specific methods in the ModifyCheckingAccountCmdImpl class**

```
...public class ModifyCheckingAccountCmdImpl extends TargetableCommandImplimplements
ModifyCheckingAccountCmd{    // Variables    ...    // Constructors    ...    // Methods in
ModifyCheckingAccountCmd interface    public float getAmount() {        return amount;    }
public float getBalance() {        return balance;    }    public float getOldBalance() {
return oldBalance;    }    public float setBalance(float amount) {        balance = balance +
amount;        return balance;    }    public float setBalance(int amount) {        balance +=
amount ;        return balance;    }    public TargetPolicy getCmdTargetPolicy() {        return
getTargetPolicy();    }    public void setCheckingAccount(CheckingAccount newCheckingAccount) {
if (checkingAccount == null) {            checkingAccount = newCheckingAccount;        }        else
System.out.println("Incorrect Checking Account (" +                newCheckingAccount + ")
specified");    }    public CheckingAccount getCheckingAccount() {        return checkingAccount;
}    ...}
```

The ModifyCheckingAccountCmd command operates on a checking account.Because commands are implemented as JavaBeans components, you manage input andoutput properties of commands using the standard JavaBeans techniques.For example, initialize input properties with set methods (likesetCheckingAccount) and retrieve output properties with get methods (likegetCheckingAccount). The get methods do not work until after thecommand's execute method has been called.

#### Implementing methods from the Command interface

The Command interface declares two methods, isReadyToCallExecute and reset,that must be implemented by the application programmer. Figure 88 shows the implementations for the ModifyCheckingAccountCmdcommand. The implementation of the isReadyToCallExecute method ensuresthat the checkingAccount variable is set. The reset method sets all ofthe variables back to starting values.

**Figure 88. Code example: Methods from the Command interface in the ModifyCheckingAccountCmdImpl class**

```
...public class ModifyCheckingAccountCmdImpl extends TargetableCommandImplimplements
ModifyCheckingAccountCmd{    ...    // Methods from the Command interface    public boolean
isReadyToCallExecute() {        if (checkingAccount != null)            return true;        else
return false;    }    public void reset() {        amount = 0;        balance = 0;        oldBalance
= 0;        checkingAccount = null;        targetPolicy = new TargetPolicyDefault();    }    ...}
```

#### Implementing methods from the TargetableCommand interface

The TargetableCommand interface declares one method, performExecute, that mustbe implemented by the application programmer. Figure 89 shows the implementation for the ModifyCheckingAccountCmdcommand. The implementation of the performExecute method does thefollowing:

- Saves the current balance (so the command can be undone by a compensatorcommand)
- Calculates the new balance

- Sets the current balance to the new balance
- Ensures that the hasOutputProperties method returns true so that thevalues are returned to the client

In addition, the ModifyCheckingAccountCmdImpl class overrides the defaultimplementation of the setOutputProperties method.

**Figure 89. Code example: Methods from the TargetableCommand interface in the ModifyCheckingAccountCmdImpl class**

```
...public class ModifyCheckingAccountCmdImpl extends TargetableCommandImplimplements
ModifyCheckingAccountCmd{    ...    // Method from the TargetableCommand interface    public void
performExecute() throws Exception {        CheckingAccount checkingAccount = getCheckingAccount();
oldBalance =  checkingAccount.getBalance();        balance = oldBalance+amount;
checkingAccount.setBalance(balance);        setHasOutputProperties(true);    }    public void
setOutputProperties(TargetableCommand fromCommand) {        try {            if (fromCommand !=
null) {               ModifyCheckingAccountCmd modifyCheckingAccountCmd =
(ModifyCheckingAccountCmd) fromCommand;                    this.oldBalance =
modifyCheckingAccountCmd.getOldBalance();               this.balance =
modifyCheckingAccountCmd.getBalance();              this.checkingAccount =
modifyCheckingAccountCmd.getCheckingAccount();                  this.amount =
modifyCheckingAccountCmd.getAmount();           }       }       catch (Exception ex) {
System.out.println("Error in setOutputProperties.");        }    }    ...}
```

#### Implementing the CompensableCommand interface

The CompensableCommand interface declares one method, getCompensatingCommand,that must be implemented by the application programmer. Figure 90 shows the implementation for the ModifyCheckingAccountCmdcommand. The implementation simply returns an instance of theModifyCheckingAccountCompensatorCmd command associated with the currentcommand.

**Figure 90. Code example: Method from the CompensableCommand interface in the ModifyCheckingAccountCmdImpl class**

```
...public class ModifyCheckingAccountCmdImpl extends TargetableCommandImplimplements
ModifyCheckingAccountCmd{    ...    // Method from CompensableCommand interface    public Command
getCompensatingCommand() throws CommandException {        modifyCheckingAccountCompensatorCmd =
new ModifyCheckingAccountCompensatorCmd(this);        return
(Command)modifyCheckingAccountCompensatorCmd;    }                                                }
```

Writing the compensating command

An application that uses a compensable command requires two separatecommands: the primary command (declared as a CompensableCommand) and thecompensating command. In the example application, the primary commandis declared in the ModifyCheckingAccountCmd interface and implemented in theModifyCheckingAccountCmdImpl class. Because this command is also acompensable command, there is a second command associated with it that isdesigned to undo its work. When you create a compensable command, youalso have to write the compensating command.

Writing a compensating command can require exactly the same steps aswriting the original command: writing the interface and providing animplementation class. In some cases, it may be simpler. Forexample, the command to compensate for the ModifyCheckingAccountCmd does notrequire any methods beyond those defined for the original command, so it doesnot need an interface. The compensating command, calledModifyCheckingAccountCompensatorCmd, simply needs to be implemented in a classthat extends the TargetableCommandImpl class. This class must:

- Provide a way to instantiate the command; the example uses aconstructor
- Implement the three required methods:
  - isReadyToCallExecute and reset--both from the Command interface
  - performExecute--from the TargetableCommand interface

Figure 91 shows the structure of the implementation class, itsvariables (references to the original command and to the relevant checkingaccount), and the constructor. The constructor simply instantiates thereferences to the primary command and account.

**Figure 91. Code example: Variables and constructor in the ModifyCheckingAccountCompensatorCmd class**

```
...public class ModifyCheckingAccountCompensatorCmd extends TargetableCommandImpl{    public
ModifyCheckingAccountCmdImpl modifyCheckingAccountCmdImpl;    public CheckingAccount
checkingAccount;       public ModifyCheckingAccountCompensatorCmd(
ModifyCheckingAccountCmdImpl originalCmd)    {        // Get an instance of the original command
modifyCheckingAccountCmdImpl = originalCmd;        // Get the relevant account
checkingAccount = originalCmd.getCheckingAccount();     }     // Methods from the Command and
Targetable Command interfaces    ....}
```

Figure 92 shows the implementation of the inherited methods.The implementation of the isReadyToCallExecute method ensures that thecheckingAccount variable has been instantiated.

The performExecute method verifies that the actual checking-account balanceis consistent with what the original command returns. If so, itreplaces the

current balance with the previously stored balance by using theModifyCheckingAccountCmd command. Finally, it saves the most-recentbalances in case the compensating command needs to be undone. The resetmethod has no work to do.

**Figure 92. Code example: Methods in ModifyCheckingAccountCompensatorCmd class**

```
...public class ModifyCheckingAccountCompensatorCmd extends TargetableCommandImpl{    // Variables
and constructor    ....   // Methods from the Command and TargetableCommand interfaces   public
boolean isReadyToCallExecute() {        if (checkingAccount != null)           return true;
else            return false;    }    public void performExecute() throws CommandException     {
try {           ModifyCheckingAccountCmdImpl originalCmd =
modifyCheckingAccountCmdImpl;           // Retrieve the checking account modified by the original
command           CheckingAccount checkingAccount = originalCmd.getCheckingAccount();           if
(modifyCheckingAccountCmdImpl.balance ==               checkingAccount.getBalance()) {
// Reset the values on the original command
checkingAccount.setBalance(originalCmd.oldBalance);                   float temp =
modifyCheckingAccountCmdImpl.balance;                originalCmd.balance =
originalCmd.oldBalance;                  originalCmd.oldBalance = temp;           }
else {               // Balances are inconsistent, so we cannot compensate             throw new
CommandException(                  "Object modified since this command ran.");          }
}        catch (Exception e) {           System.out.println(e.getMessage());       }    }
public void reset() {}}
```

## Using a command

To use a command, the client creates an instance of the command and callsthe command's execute method. Depending on the command, callingother methods can be necessary. The specifics will vary with theapplication.

In the example application, the server is the CheckingAccountBean, anentity enterprise bean. In order to use this enterprise bean, theclient gets a reference to the bean's home interface. The clientthen uses the reference to the home interface and one of the bean'sfinder methods to obtain a reference to the bean's remoteinterface. If there is no appropriate bean, the client can create oneusing a create method on the home interface. All of this work isstandard enterprise bean programming covered elsewhere in thisdocument.

Figure 93 illustrates the use of the ModifyCheckingAccountCmdcommand. This work takes place after an appropriate CheckingAccountbean has been found or created. The code instantiates a command,setting the input values by using one of the constructors defined for thecommand. The null argument indicates that the command should look upthe server using the default target policy, and 1000 is the amount the commandattempts to add to the balance of the checking account. (For moreinformation on how the command package uses defaults to determine the targetof a command, see The default target policy.) After the command is instantiated, the code callsthe setCheckingAccount method to identify the account to be modified.Finally, the execute method on the command is called.

**Figure 93. Code example: Using the ModifyCheckingAccountCmd command**

```
{    ...    CheckingAccount checkingAccount    ...    try {       ModifyCheckingAccountCmd cmd =
new ModifyCheckingAccountCmdImpl(null, 1000);       cmd.setCheckingAccount(checkingAccount);
cmd.execute();    }    catch (Exception e) {      System.out.println(e.getMessage());    }    ...}
```

#### Using a compensating command

To use a compensating command, you must retrieve the compensator associatedwith the primary command and call its execute method. Figure 94 shows the code used to run the original command and to givethe user the option of undoing the work by running the compensatingcommand.

**Figure 94. Code example: Using the ModifyCheckingAccountCompensator command**

```
{    ...    CheckingAccount checkingAccount    ....    try {       ModifyCheckingAccountCmd cmd =
new ModifyCheckingAccountCmdImpl(null, 1000);       cmd.setCheckingAccount(checkingAccount);
cmd.execute();       ...       System.out.println("Would you like to undo this work?  Enter Y or
N");       try {           // Retrieve and validate user's response           ...       }
...       if (answer.equalsIgnoreCase(Y)) {           Command compensatingCommand =
cmd.getCompensatingCommand();           compensatingCommand.execute();       }    }    catch
(Exception e) {       System.out.println(e.getMessage());    }    ...}
```

## Writing a command target (server)

In order to accept commands, a server must implement the CommandTargetinterface and its single method, executeCommand.

The example application implements the CommandTarget interface in anenterprise bean. (For a servlet-based example, see Writing a command target (client-side adapter).) The target enterprise bean can be a session bean oran entity bean. You can write a target enterprise bean that forwardscommands to a specific server, such as another entity bean. In thiscase, all commands directed at a specific target go through the targetenterprise bean. You can also write a target enterprise bean that doesthe work of the command locally.

Make an enterprise bean the target of a command by:

- Extending the CommandTarget interface when you define the bean'sremote interface, which must also extend the EJBObject interface
- Implementing the CommandTarget interface when you implement the beanclass, which must also implement either the SessionBean or EntityBeaninterface

The target of the example application is an enterprise bean calledCheckingAccountBean. This bean's remote interface,CheckingAccount, extends the CommandTarget interface in addition to theEJBObject interface. The methods declared in the remote interface areindependent of those used by the command. The executeCommand isdeclared in neither the bean's home nor remote interfaces. Figure 95 shows the CheckingAccount interface.

**Figure 95. Code example: The remote interface for the CheckingAccount entity bean, also a command target**

```
...import com.ibm.websphere.command.*;import javax.ejb.EJBObject;import
java.rmi.RemoteException;public interface CheckingAccount extends CommandTarget, EJBObject {
float deposit (float amount) throws RemoteException;    float deposit (int amount) throws
RemoteException;    String getAccountName() throws RemoteException;    float getBalance() throws
RemoteException;    float setBalance(float amount) throws RemoteException;    float withdrawal
(float amount) throws RemoteException, Exception;    float withdrawal (int amount) throws
RemoteException, Exception;}
```

The enterprise bean class, CheckingAccountBean, implements the EntityBeaninterface as well as the CommandTarget interface. The class containsthe business logic for the methods in the remote interface, the necessarylife-cycle methods (ejbActivate, ejbStore, and so on), and the executeCommanddeclared by the CommandTarget interface. The executeCommand method isthe only command-specific code in the enterprise bean class. Itattempts to run the performExecute method on the command and throws aCommandException if an error occurs. If the performExecute method runssuccessfully, the executeCommand method uses the hasOutputProperties method todetermine if there are output properties that must be returned. If thecommand has output properties, the method returns the command object to theclient. Figure 96 shows the relevant parts of the CheckingAccountBeanclass.

**Figure 96. Code example: The bean class for the CheckingAccount entity bean, also a command target**

```
...public class CheckingAccountBean implements EntityBean, CommandTarget {    // Bean variables
...    // Business methods from remote interface    ...    // Life-cycle methods for CMP entity
beans    ...    // Method from the CommandTarget interface    public TargetableCommand
executeCommand(TargetableCommand command)    throws RemoteException, CommandException    {
try {            command.performExecute();        }        catch (Exception ex) {            if (ex
instanceof RemoteException) {                RemoveException remoteException = (RemoteException)ex;
if (remoteException.detail != null) {                    throw new
CommandException(remoteException.detail);                    }                throw new
CommandException(ex);            }        }        if (command.hasOutputProperties()) {
return command;        }        return null;    }}
```

## Targets and target policies

A targetable command extends the TargetableCommand interface, which allows theclient to direct a command to a particular server. TheTargetableCommand interface (and the TargetableCommandImpl class) provide twoways for a client to specify a target: the setCommandTarget andsetCommandTargetName methods. (These methods were introduced in The TargetableCommand interface.) The setCommandTarget methods allows the client toset the target object directly on the command. The setCommandTargetNamemethod allows the client to refer to the server by name; this approach isuseful when the client is not directly aware of server objects. Atargetable command also has corresponding getCommandTarget andgetCommandTargetName methods.

The command package needs to be able to identify the target of acommand. Because there is more than one way to specify the target andbecause different applications can have different requirements, the commandpackage does not specify a selection algorithm. Instead, it provides aTargetPolicy interface with one method, getCommandTarget, and a defaultimplementation. This allows applications to devise custom algorithmsfor determining the target of a command when appropriate.

#### The default target policy

The command package provides a default implementation of the TargetPolicyinterface in the TargetPolicyDefault class. If you use this defaultimplementation, the command determines the target by looking through anordered sequence of four options:

1. The CommandTarget value
2. The CommandTargetName value
3. A registered mapping of a target for a specific command
4. A defined default target

If it finds no target, it returns null.The TargetPolicyDefault class provides methods for managing the assignment ofcommands with targets (registerCommand, unregisterCommand, and listMappings),and a method for setting a default name for the target(setDefaultTargetName). The default target name iscom.ibm.websphere.command.LocalTarget, whereLocalTarget is a class that runs the command's performExecute methodlocally. Figure 97 shows the relevant variables and the methods in theTargetPolicyDefault class.

**Figure 97. Code example: The TargetPolicyDefault class**

```
...public class TargetPolicyDefault implements TargetPolicy, Serializable{    ...    protected
String defaultTargetName = "com.ibm.websphere.command.LocalTarget";    public CommandTarget
getCommandTarget(TargetableCommand command) {        ... }    public Dictionary listMappings() {
... }    public void registerCommand(String commandName, String targetName) {        ... }    public
void unregisterCommand(String commandName) {        ... }    public void seDefaultTargetName(String
defaultTargetName) {        ... }}
```

Setting the command targetThe ModifyCheckingAccountImpl class provides two command constructors (see Figure 86). One of them takes a command target as an argumentand implicitly uses the default target policy to locate the target. Theconstructor used in Figure 93 passes a null target, so that the default target policytraverses its choices and eventually finds the default target name,LocalTarget.

The example in Figure 98 uses the same constructor to set the targetexplicitly. This example differs from Figure 93 as follows:

- The command target is set to the checking account rather than null.The default target policy starts to traverse its choices and finds the targetin the first place it looks.
- It does not have to call the setCheckingAccount method to indicate theaccount on which the command should operate; the constructor uses thetarget variable as both the target and the account.

**Figure 98. Code example: Identifying a target with CommandTarget**

```
{    ...    CheckingAccount checkingAccount    ....    try {        ModifyCheckingAccountCmd cmd =
new ModifyCheckingAccountCmdImpl(checkingAccount, 1000);        cmd.execute();    }    catch
(Exception e) {       System.out.println(e.getMessage());    }    ...}
```

Setting the command target nameIf a client needs to set the target of the command by name, it can use thecommand's setCommandTargetName method. Figure 99 illustrates this technique. This example compareswith Figure 93 as follows:

- Both explicitly set the command target in the constructor to null.
- Both use the setCheckingAccount method to indicate the account on whichthe command should operate.
- This example sets the target name explicitly by using thesetCommandTargetName method. When the default target policy traversesits choices, it finds a null for the first choice and a name for thesecond.

**Figure 99. Code example: Identifying a target with CommandTargetName**

```
{    ...    CheckingAccount checkingAccount    ....    try {        ModifyCheckingAccountCmd cmd =
new ModifyCheckingAccountCmdImpl(null, 1000);       cmd.setCheckingAccount(checkingAccount);
cmd.setCommandTargetName("com.ibm.sfc.cmd.test.CheckingAccountBean");        cmd.execute();    }
catch (Exception e) {       System.out.println(e.getMessage());    }    ...}
```

Mapping the command to a target nameThe default target policy also permits commands to be registered withtargets. Mapping a command to a target is an administrative task thatmost appropriately done through a configuration tool. The WebSphereApplication Server administrative console does not yet support theconfiguration of mappings between commands and targets. Applicationsthat require support for the registration of commands with targets must supplythe tools to manage the mappings. These tools can be visual interfacesor command-line tools.

Figure 100 shows the registration of a command with a target.The names of the command class and the target are explicit in the code, but inpractice, these values would come from fields in a user interface or argumentsto a command-line tool. If a program creates a command as shown in Figure 93, with a null for the target, when the default target policytraverses its choices, it finds a null for the first and second choices and amapping for the third.

**Figure 100. Code example: Mapping a command to a target in an external application**

```
{    ...    targetPolicy.registerCommand(        "com.ibm.sfc.cmd.test.ModifyCheckingAccountImpl",
"com.ibm.sfc.cmd.test.CheckingAccountBean");    ...}
```

#### Customizing target policies

You can define custom target policies by implementing the TargetPolicyinterface and providing a getCommandTarget method appropriate for yourapplication. The TargetableCommandImpl class provides setTargetPolicyand getTargetPolicy methods for managing custom target policies.

So far, the target of all the commands has been a checking-account entitybean. Suppose that someone introduces a session enterprise bean(MySessionBean) that can also act as a command target. Figure 101 shows a simple custom policy that sets the target of everycommand to MySessionBean.

**Figure 101. Code example: Creating a custom target policy**

```
...import java.io.*;import java.util.*;import java.beans.*;import com.ibm.websphere.command.*;public
class CustomTargetPolicy implements TargetPolicy, Serializable {    public CustomTargetPolicy {
super();    }    public CommandTarget getCommandTarget(TargetableCommand command) {
CommandTarget = null;        try {            target = (CommandTarget)Beans.instantiate(null,
"com.ibm.sfc.cmd.test.MySessionBean");        }        catch (Exception e) {
e.printStackTrace();        }    }}
```

Since commands are implemented as JavaBeans components, using custom targetpolicies requires importing the java.beans package and writing someelementary JavaBeans code. Also, your custom target-policy class mustalso implement the java.io.Serializable interface.

Using a custom target policyThe ModifyCheckingAccountImpl class provides two command constructors (see Figure 86). One of them implicitly uses the default targetpolicy; the other takes a target policy object as an argument, whichallows you to use a custom target policy. The example in Figure 102 uses the second constructor, passing a null target and acustom target policy, so that the custom policy is used to determine thetarget. After the command is executed, the code uses the reset methodto return the target policy to the default.

**Figure 102. Code example: Using a custom target policy**

```
{    ...    CheckingAccount checkingAccount    ....    try {        CustomTargetPolicy customPolicy
= new CustomTargetPolicy();        ModifyCheckingAccountCmd cmd =            new
ModifyCheckingAccountCmdImpl(null, 1000, customPolicy);
cmd.setCheckingAccount(checkingAccount);        cmd.execute();        cmd.reset();    }    catch
(Exception e) {       System.out.println(e.getMessage());    }}
```

## Writing a command target (client-side adapter)

Commands can be used with any Java application, but the means of sending thecommand from the client to the server varies. The application describedin The example application used enterprise beans. The example in this sectionshows how you can send a command to a servlet over the HTTP protocol.

In this example, the client implements the CommandTarget interfacelocally. Figure 103 shows the structure of the client-side class;it implements the CommandTarget interface by implementing the executeCommandmethod.

**Figure 103. Code example: The structure of a client-side adapter for a target**

```
...import java.io.*;import java.rmi.*;import com.ibm.websphere.command.*;public class
ServletCommandTarget implements CommandTarget, Serializable{    protected String hostName =
"localhost";  public static void main(String args[]) throws Exception    {        ....    }
public TargetableCommand executeCommand(TargetableCommand command)        throws CommandException
{        ....    }      public static final byte[] serialize(Serializable serializable)
throws IOException {        ...  }    public String getHostName() {        ... }    public void
setHostName(String hostName) {        ... }    private static void showHelp() {        ... }}
```

The main method in the client-side adapter constructs and intializes theCommandTarget object, as shown in Figure 104.

**Figure 104. Code example: Instantiating the client-side adapter**

```
public static void main(String args[]) throws Exception{    String hostName =
InetAddress.getLocalHost().getHostName();    String fileName = "MyServletCommandTarget.ser";    //
Parse the command line    ...    // Create and initialize the client-side CommandTarget adapter
ServletCommandTarget servletCommandTarget = new ServletCommandTarget();
servletCommandTarget.setHostName(hostName);    ...    // Flush and close output streams    ...}
```

#### Implementing a client-side adapter

The CommandTarget interface declares one method, executeCommand, which theclient implements. The executeCommand method takes a TargetableCommandobject as input; it also returns a TargetableCommand. Figure 105 shows the implementation of the method used in theclient-side adapter. This implementation does the following:

- Serializes the command it receives
- Creates an HTTP connection to the servlet
- Creates input and output streams, to handle the command as it is sent tothe server and returned
- Places the command on the output stream
- Sends the command to the server
- Retrieves the returned command from the input stream
- Returns the returned command to the caller of the executeCommand method

**Figure 105. Code example: A client-side implementation of the executeCommand method**

```
public TargetableCommand executeCommand(TargetableCommand command)     throws CommandException{
try {         // Serialize the command        byte[] array = serialize(command);        // Create a
connection to the servlet        URL url = new URL           ("http://" + hostName +
"/servlet/com.ibm.websphere.command.servlet.CommandServlet");        HttpURLConnection
httpURLConnection =            (HttpURLConnection) url.openConnection();        // Set the
properties of the connection        ...        // Put the serialized command on the output stream
OutputStream outputStream = httpURLConnection.getOutputStream();        outputStream.write(array);
// Create a return stream        InputStream inputStream = httpURLConnection.getInputStream();
// Send the command to the servlet        httpURLConnection.connect();        ObjectInputStream
objectInputStream =            new ObjectInputStream(inputStream);        // Retrieve the command
returned from the servlet        Object object = objectInputStream.readObject();        if (object
instanceof CommandException) {            throw ((CommandException) object);        }        // Pass
the returned command back to the calling method        return (TargetableCommand) object;    }    //
Handle exceptions    ....}
```

#### Running the command in the servlet

The servlet that runs the command is shown in Figure 106. The service method retrieves the command from theinput stream and runs the performExecute method on the command. Theresulting object, with any output properties that must be returned to theclient, is placed on the output stream and sent back to the client.

**Figure 106. Code example: Running the command in the servlet**

```
...import java.io.*;import javax.servlet.*;import javax.servlet.http.*;import
com.ibm.websphere.command.*;public class CommandServlet extends HttpServlet {    ...    public void
service(HttpServletRequest request,                        HttpServletResponse response)
throws ServletException, IOException     {        try {            ...            // Create input
and output streams            InputStream inputStream = request.getInputStream();
OutputStream outputStream = response.getOutputStream();           // Retrieve the command from the
input stream            ObjectInputStream objectInputStream =                 new
ObjectInputStream(inputStream);           TargetableCommand command = (TargetableCommand)
objectInputStream.readObject();           // Create the command for the return stream
Object returnObject = command;             // Try to run the command's performExecute method
try {                command.performExecute();                  }           // Handle exceptions
from the performExecute method            ...              // Return the command with any output
properties            ObjectOutputStream objectOutputStream =                 new
ObjectOutputStream(outputStream);           objectOutputStream.writeObject(returnObject);
// Flush and close output streams           ...        }        catch (Exception ex) {
ex.printStackTrace();        }    }}
```

In this example, the target invokes the performExecute method on thecommand, but this is not always necessary. In some applications, it canbe preferable to implement the work of the command locally. Forexample, the command can be used only to send input data, so that the targetretrieves the data from the command and runs a local database procedure basedon the input. You must decide the appropriate way to use commands inyour application.

---

## The localizable-text package

### Overview

Users of distributed applications can come from widely varying areas;they can speak different languages, represent dates and times in regionallyspecific ways, and use different currencies. An application intended tobe used by such an audience must either force them all to use the sameinterface (for example, an English-based interface), or it can be written insuch a way that it can be configured to the linguistic conventions of theusers, so English-speaking users can use the English interface butFrench-speaking users can interact with the application through a Frenchinterface.

An application that can present information to users in formats that abideby the users' linguistic conventions is said to be*localizable:* the application can be configured to interactwith users from different localities in linguistically appropriateways. In a localized application, a user in one region sees errormessages, output, and interface elements (like menu options) in the requestedlanguage. Additionally, other elements that are not strictlylinguistic, like date and time formats and currencies, are presented in theappropriate style for users in the specified region. A user in anotherregion sees output in the conventional language or format for thatregion.

Historically, the creation of localizable applications has been restrictedto large corporations writing complex systems. The strategies forwriting localizable code, collectively called *internationalizationtechniques*, have traditionally been expensive and difficult toimplement, so they have been applied only to major development efforts.However, given the rise in distributed computing and in use of the World WideWeb, application developers have been pressured to make a much wider varietyof applications localizable. This requires makinginternationalization--the techniques for writing localizableprograms--much more accessible to application developers. TheWebSphere localizable-text package is a set of Java classes and interfacesthat can be used by WebSphere application developers to localize distributedWebSphere applications easily. Language catalogs for distributedWebSphere applications can be stored centrally, so the catalogs can bemaintained and administered efficiently.

In a nonlocalizable application, parts of the application that a user seesare unalterably coded into the application. For example, a routine thatprints an error message simply prints a string, probably in English, to a fileor the console. A localizable program adds a layer of abstraction intothe design. Instead of going simply from error condition to outputstring, a localizable program represents error messages with somelanguage-neutral information; in the simplest case, each error conditioncorresponds to a key. In order to print a usable error string for theuser, the application looks up the key in the configured messagecatalog. A message catalog is a list of keys with correspondingstrings. Different message catalogs provide the strings in differentlanguages. The application looks up the key in the appropriate catalog,retrieves the corresponding error message in the desired language, and printsthis string for the user.

The technique of localization can be used for far more than translatingerror messages. For example, by using keys to represent eachelement--button, label, menu item, and so forth--in a graphical userinterface and by providing a message catalog containing translations of thebutton names, labels, and menu items, the graphical interface can beautomatically translated into multiple languages. In addition,extending support to additional languages requires providing message catalogsfor those languages; the application itself requires nomodification.

Localization of an application is driven by two variables, the time zoneand the locale. The time zone variable indicates how to compute thelocal time as an offset from a standard time like Greenwich Mean Time.The locale is a collection of information that indicates a geographic,political, or cultural region. It provides information on language,currency, and the conventions for presenting information like dates, and in alocalizable program, the locale also indicates the message catalog from whichan application retrieves messages. A time zone can cover many locales,and a single locale can span time zones. With both time zone andlocale, the date, time, currency, and language for users in a specific regioncan be determined.

Identifying localizable text

To write a localizable application, an application developer must determinewhich aspects of the application need to be translatable. These aretypically the parts of an application a user must read and understand.Application developers must consider the parts of an application with whichall users directly interact, like the application's interface, and theparts serving more specialized purposes, like messages in log files.Good candidates for localization include:

- Elements in graphical user interfaces
  - Title bars for windows
  - Menu names, and the items on the menus (for example, "select File >Open")
  - Labels on buttons (for example, "click the OK button")
  - Instructions directing users to fill in fields (for example, "enter theaccount number")
  - Any other elements that users must read
- Prompts in command-line interfaces
- Output from the program
  - Responses to user input
  - Error messages
  - Text returned when exceptions are thrown
  - Other status messages (warnings, audit messages, and others)

After identifying each element of the application to be localized,application developers must assign a unique key to each element and provide amessage catalog for each language to be supported. Each message catalogconsists of keys and the corresponding language-specific strings. Thekey, therefore, is the link between the program and the message catalog;the program internally refers to localizable elements by key and uses themessage catalog to generate the output seen by the user. Translatedstrings are generated by calling the format method on aLocalizableTextFormatter object, which represents a key and a resource bundle(a set of message catalogs). The locale setting of the programdetermines the message catalog in which to search for the key.

Creating message catalogs

After identifying each element to be localized, message catalogs must becreated for each language to be supported. These catalogs, which areimplemented as Java resource bundles, can be created in two ways, either assubclasses of the ResourceBundle class or as Java properties files.Resource bundles have a variety of uses in Java; for message catalogs,the properties-file approach is more common. If properties files areused, support for languages to be added or removed without modifying theapplication code, and catalogs can be prepared by people without programmingexpertise.

A message catalog implemented in a properties file consists of a line foreach key, where a key identifies a localizable element. Each line inthe file has the following structure:

```
key = String corresponding to the key
```

For example, a grapical user interface for a banking system can have apull-down menu to be used for selecting a type of account, like savings orchecking. The label for the pull-down menu and the account types on themenu are good choices for localization. There are three elements thatrequire keys: the label for the account menu and the two items on themenu. If the keys are accountString, savingsString, and checkingString,the English properties file associates each with an English string.

**Figure 107. Three elements in an English message catalog**

```
accountString = AccountssavingsString = SavingscheckingString = Checking...
```

In the German properties files, each key is given a corresponding Germanvalue.

**Figure 108. Three elements in a German message catalog**

```
accountString = KontensavingsString = SparkontocheckingString = Girokonto ...
```

Properties files can be added for any other needed languages, aswell.

#### Naming the properties files

To enable resolution to a specific properties file, Java specifies namingconventions for the properties files in a resource bundle:`resourceBundleName_localeID.properties`

Each file takes a fixed extension, `.properties`.The set of files making up the resource bundle is given a collectivename; for a simple banking application, an obvious name, likeBankingResources, suffices for the resource bundle. Each file is giventhe name of the resource bundle with a locale identifier; the specificvalue of the locale ID varies with the locale. These are usedinternally by the Java.util.ResourceBundle class to match filesin a resource bundle to combinations of locale and time-zone settings.The details of the algorithm vary with the release of the JDK; see yourJava documentation for information specific to your installation.

In the banking application, typical files in the BankingResources resourcebundle include BankingResources_en.properties for the English messagecatalog and BankingResources_de.properties for the Germancatalog. Additionally, a default catalog,BankingResources.properties, is provided for use when the requestedcatalog cannot be found. The default catalog is often theEnglish-language catalog.

Resource bundles containing message catalogs for use with localizable textneed to be installed only on the systems where the formatting of strings isactually performed. The resource bundles are typically placed in anapplication's JAR file. See WebSphere support for more information.

#### Localization support in WebSphere and Java

The Java packagecom.ibm.websphere.i18n.localizabletext containsthe classes and interfaces constituting the localizable-text package.This package makes extensive use of the internationalization and localizationfeatures of the Java language; programmers using the WebSpherelocalizable-text package must understand the underlying Java support, whichare not documented in any detail here.

Java support

The WebSphere localizable-text package relies primarily on the followingJava components:

- java.util.Locale
- java.util.TimeZone
- java.util.ResourceBundle
- java.text.MessageFormat

This list is not exhaustive. WebSphere and these Java classes canalso use related Java classes, but the related classes--for example,java.util.Calendar--are typically special-purposesclasses. This section briefly describes only the primaryclasses.

#### Locale

A Locale object in Java encapsulates a language and a geographic region,for example, the java.util.Locale.US object containslocale information for the United States. An application that specifiesa locale can then take advantage of the locale-sensitive formatters built intothe Java language. These formatters, in the java.text package,handle the presentation of numbers, currency values, dates, and times.

#### TimeZone

A TimeZone object in Java encapsulates a representation of the time andprovides methods for tasks like reporting the time and accommodating seasonaltime shifts. Applications use the time zone to determine the local dateand time.

#### ResourceBundle

A resource bundle is a named collection of resources--information usedby the application, for example, strings, fonts, and images--used by aspecific locale. The ResourceBundle class allows an application toretrieve the named resource bundle appropriate to the locale. Resourcebundles are used to hold the messages catalogs, as described in Writing localizable programs. Resource bundles can be implemented in two ways,either as subclasses of the ResourceBundle class or as Java propertiesfiles.

#### MessageFormat

The MessageFormat class can be used to construct strings based onparameters. As a simple example, suppose a localized applicationrepresents a particular error condition with a numeric key. When theapplication reports the error condition, it uses a message formatter toconvert the numeric key into a meaningful string. The message formatterconstructs the output string by looking up the code (the parameter) in anappropriate resource bundle and retrieving the corresponding string from themessage catalog. Additional parameters--for example, another keyrepresenting the program module--can also be used in assembling theoutput message.

WebSphere support

The WebSphere localizable-text package wraps the Java support and extendsit for efficient and simple use in a distributed environment. Theprimary

class used by application programmers is the LocalizableTextFormatterclass. Objects of this class are created, typically in server programs,but clients can also create them. LocalizableTextFormatter objects arecreated for specific resource-bundle names and keys. Client programsthat receive LocalizableTextFormatter objects call the object's formatmethod. This method uses the locale of the client application toretrieve the appropriate resource bundle and assemble the locale-specificmessage based on the key.

For example, suppose that a WebSphere client-server application supportsboth French and English locales; the server is using an English localeand the client, a French locale. The server creates two resourcebundles, one for English and one for French. When the client makes arequest that triggers a message, the server creates a LocalizableTextFormatterobject containing the name of the resource bundle and the key for the message,and passes the object back to the client.

When the client receives the LocalizableTextFormatter object, it calls theobject's format method, which returns the message corresponding to thekey from the French resource bundle. The format method retrieves theclient's locale and, using the locale and name of the resource bundle,determines the resource bundle corresponding to the locale. (If theclient has set an English locale, calling the format method results in theretrieval of an English message.) The formatting of the message istransparent to the client.In this simple client-server example, the resource bundles reside centrallywith the server. The client machine does not have to installthem. Part of what the WebSphere localizable-text package provides isthe infrastructure to support centralized catalogs. WebSphere uses anenterprise bean, a stateless session bean provided with the localizable-textpackage, to access the message catalogs. When the client calls theformat method on the LocalizableTextFormatter object, the following eventsoccur internally:

1. The client application sets the time zone and locale values in theLocalizableTextFormatter object, either by passing them explicitly or throughdefaults.
2. A call, LocalizableTextFormatterEJBFinder, is made to retrieve a referenceto the formatting enterprise bean.
3. Information from the LocalizableTextFormatter object, including theclient's time zone and locale, is sent to the formatting bean.
4. The formatting bean uses the name of the resource bundle, the message key,the time zone, and the locale to assemble the language-specificmessage.
5. The enterprise bean returns the formatted message to the client.
6. The formatted message is inserted into the LocalizableTextFormatter objectand returned by the format method.

A call to a LocalizableTextFormatter.format method requires at most oneremote invocation, to contact the formatting enterprise bean. However,the LocalizableTextFormatter object can optionally cache formatted messages,eliminating the formatting call for subsequent uses. It also allows theapplication to set a fallback string; this means the application canstill return a readable string even if it cannot access a message catalog toretrieve the language-specific string. Additionally, the resourcebundles can be stored locally. The localizable-text package provides astatic variable that indicates whether the bundles are stored locally(LocalizableConfiguration.LOCAL) or remotely(LocalizableConfiguration.REMOTE), but the setting of this variableapplies to all applications running within a Java Virtual Machine(JVM).

The LocalizableTextFormatter class

The LocalizableTextFormatter class, found in the packagecom.ibm.websphere.i18n.localizabletext, is theprimary programming interface for using the localizable-text package.Objects of this class contain the information needed to createlanguage-specific strings from keys and resource bundles.

#### Location of message catalogs and the ApplicationName value

Applications written with the WebSphere localizable-text package can storemessage catalogs locally or remotely. In a distributed environment, theuse of remote, centrally stored catalogs is appropriate. Allapplications can use the same catalogs, and administration and maintenance ofthe catalogs are simplified; each component does not need to store andmaintain copies of the message catalogs. Local formatting is useful intest situations and appropriate under some circumstances. In order tosupport both local and remote formatting, a LocalizableTextFormatter objectmust indicate the name of the formatting application. For example, whenan application formats a message by using remote, centrally stored catalogs,the message is actually formatted by a simple enterprise bean (see WebSphere support for more information). Although the localizable-textpackage contains the code to automate looking up the enterprise bean andissuing a call to it, the application needs to know the name of the formattingenterprise bean. Several methods in the LocalizableTextFormatter classuse a value described as *application name*; this refers to thename of the formatting application, which is not necessarily the name of theapplication in which the value is set.

#### Caching messages

The LocalizableTextFomatter object can optionally cache formatted messagesso that they do not have to be reformatted when needed again. Bydefault, caching is not used, but theLocalizableTextFormatter.setCacheSetting method can be used to enablecaching. When caching is enabled and theLocalizableTextFormatter.format method is called, the method determineswhether the message has already been formatted. If so, the cachedmessage is returned. If the message is not found in the cache, themessage is formatted and returned to the caller, and a copy of the message iscached for future use.

If caching is disabled after messages have been cached, those messagesremain in the cache until the cache is cleared by a call to theLocalizableTextFormatter.clearCache method. The cache can becleared at any time. The cache within a LocalizableTextFormatter objectis automatically cleared when any of the following methods are called on theobject:

- setResourceBundleName(String resourceBundleName)
- setPatternKey(String patternKey)
- setArguments(Object[] args)
- setApplicationName(String appName)

#### Fallback information

Under some circumstances, it can be impossible to format a message.The localizable-text package implements a fallback strategy, making itpossible to get some information even if a message cannot be correctlyformatted into the desired language. The LocalizableTextFomatter objectcan optionally store a fallback value for a message string, the time zone, andthe locale. These can be ignored unless the LocalizableTextFormatterobject throws an

exception.

#### Application-specific variables

The localizable-text package provides native support for localization basedon time zone and locale, but application developers can construct messages onthe basis of other values as well. The localizable-text packageprovides an illustrative class, LocalizableTextDateTimeArgument, which reportsthe date and time. The date and time information is localized by usingthe locale and time-zone values, but the class also uses additional variablesto determine how the output is presented. The date and time informationcan be requested in a variety of styles, from the fully detailed to theterse. In this example, the construction of message strings is drivenby three variables: the locale, the time zone, and the style.Applications can use any number of variables in addition to locale and timezone for constructing messages. See Using optional arguments for more information.

## Writing a localizable application

To develop a WebSphere application that uses localizable text, applicationdevelopers must do the following:

- Determine the parts of the application to be localized.
  - Identify the application elements to be localized and assign each akey.
  - Create message catalogs for each language by associating a string witheach key.

  These tasks were described previously. See Identifying localizable text and Creating message catalogs for more information.
- Assemble language-specific strings from keys, resource bundles, and otherarguments.
  - Create a LocalizableTextFormatter object.
  - Set the values within the object for the key, the name of the resourcebundle, the name of the remote formatting application, and any optionalarguments.
  - Call the format method on the LocalizableTextObject, which returns theassembled string.

  This section describes these tasks.

#### Creating a LocalizableTextFormatter object

Server programs typically create LocalizableTextFormatter objects, whichare sent to clients as the result of some operation; clients format theobjects at the appropriate time. Less typically, clients can createLocalizableTextFormatter objects locally. To create aLocalizableTextFormatter object, applications use one of the constructors inthe LocalizableTextFormatter class:

- LocalizableTextFormatter()
- LocalizableTextFormatter(String resourceBundleName, String patternKey,String appName)
- LocalizableTextFormatter(String resourceBundleName, String patternKey,String appName, Object[] args)

The LocalizableTextFormatter object must have values set for the name ofthe resource bundle, the key, the name of the formatting application, and forany optional values so the object can be formatted. TheLocalizableTextFormatter object can be created and the values set in one stepby using the constructor that takes the necessary arguments, or the object canbe created and the values set in separate steps. Values are set byusing methods on the LocalizableTextFormatter object; for setting thevalues manually, rather than by using a constructor, use these methods:

- setResourceBundleName(String resourceBundleName)
- setPatternKey(String patternKey)
- setApplicationName(String appName)
- setArguments(Object[] args)

**Note:**

When values in the array of optional arguments are set within aLocalizableTextFormatter object, they are copied into the object, notreferenced. If an array variable holding a value is changed after thevalue has been copied into the LocalizableTextFormatter object, the value inthe LocalizableTextFormatter object will not reflect the change unless it isalso reset.

A LocalizableTextFormatter object also has methods that can be used to setvalues that cannot be set when the object is created, for example:

- To toggle the cache setting for the LocalizableTextFormatter object, usethe setCacheSetting(boolean setting) method (See Caching messages for more information.)
- To clear the cache, use the clearLocalizableTextFormatter method
- To set fallback values, use these methods:
  - setFallBackString
  - setFallBackLocale
  - setFallBackTimeZone

  (See Fallback information for more information.)

Each of these set methods also has a corresponding get method forretrieving the value. The clearLocalizableTextFormatter method unsetsall values, returning the LocalizableTextFormatter object to a blankstate. After clearing the object, reuse the object by setting newvalues and calling the format method again.

Figure 109 creates a LocalizableTextFormatter object by using thedefault constructor and uses methods on the new object to set values for thekey,

name of the resource bundle, name of the formatting application, andfallback string on the object.

**Figure 109. Code example: Creating a LocalizableTextFormatter object and setting values on it**

```
import com.ibm.websphere.i18n.localizabletext.LocalizableException;import
com.ibm.websphere.i18n.localizabletext.LocalizableTextFormatter;import java.util.Locale;public void
drawAccountNumberGUI(String accountType) {  ...  LocalizableTextFormatter ltf = new
LocalizableTextFormatter();  ltf.setPatternKey("accountNumber");
ltf.setResourceBundleName("BankingSample.BankingResources");
ltf.setApplicationName("BankingSample");  ltf.setFallBackString("Enter account number: ");  ...}
```

#### Setting localization values

The application requesting a localized message can specify the locale andtime zone for which the message is to be formatted, or the application can usethe default values set for the JVM. For example, a graphical userinterface can allow users to select the language in which to display themenus. A default value must be set, either in the environment orprogrammatically, so the menus can be generated when the application firststarts, but users can then change the menu language to suit theirneeds. Figure 110 illustrates how to change the locale used by theapplication based on the selection of a menu item.

**Figure 110. Code example: Setting the locale programmatically**

```
import java.awt.event.ActionListener;import java.awt.event.ActionEvent;...import
java.util.Locale;public void actionPerformed(ActionEvent event) {  String action =
event.getActionCommand();  ...   if (action.equals("en_us")) {       applicationLocale = new
Locale("en", "US");     ...  }  else if (action.equals("de_de")) {     applicationLocale = new
Locale("de", "DE");     ...  }  else if (action.equals("fr_fr")) {     applicationLocale = new
Locale("fr", "FR");     ...  }  ...}
```

When an application calls a format method, it can specify no arguments,which causes the message to be formatted using the JVM's default valuesfor locale and time zone, or a combination of locale and time zone can bespecified to override the JVM's defaults. (See Generating the localized text for more information on the arguments to the formatmethods.)

#### Generating the localized text

After the LocalizableTextFormatter object has been created and theappropriate values set, the object can be formatted to generate the stringappropriate to the locale and time zone. The format methods in theLocalizableTextFormatter class perform the work necessary to generate a stringfrom a set of message keys and resource bundles, based on locale and timezone. The LocalizableTextFormatter class provides four formatmethods. Each format method returns the formatted messagestring. The methods take a combination ofjava.util.Locale and java.util.TimeZone objectsand throw LocalizableException objects:

- String format();
- String format(locale);
- String format(timeZone);
- String format(locale, timeZone);

The format method with no arguments uses the locale and time-zone valuesset as defaults for the JVM. The other format methods can be used tooverride either or both of these values.

Figure 111 shows the creation of a localized string for theLocalizableTextFormatter object created in Figure 109; formatting is based on the locale set in Figure 110. If the formatting fails, theapplication retrieves and uses the fallback string instead of the localizedstring.

**Figure 111. Code example: Formatting a LocalizableTextFormatter object**

```
import com.ibm.websphere.i18n.localizabletext.LocalizableException;import
com.ibm.websphere.i18n.localizabletext.LocalizableTextFormatter;import java.util.Locale;public void
drawAccountNumberGUI(String accountType) {  ...  LocalizableTextFormatter ltf = new
LocalizableTextFormatter();  ltf.setPatternKey("accountNumber");
ltf.setResourceBundleName("BankingSample.BankingResources");
ltf.setApplicationName("BankingSample");  ltf.setFallBackString("Enter account number: ");  try {
msg = new Label (ltf.format(this.applicationLocale) , Label.CENTER);  }  catch (LocalizableException
le) {     msg = new Label(ltf.getFallBackString(), Label.CENTER);  }  ...}
```

## Using optional arguments

The localizable-text package allows users to specify an array of optionalarguments in a LocalizableTextFormatter object. These optionalarguments can greatly enhance the kinds of localization done in WebSphereapplications. This section describes two ways in which applications canuse the optional arguments:

- To assemble and format complex strings with variable substrings

- To customize the formatting of strings, taking variables other than localeand time zone into account

#### Assembling complex strings

All of the keys discussed so far have represented flat strings; duringlocalization, a string in the appropriate language is substituted for thekey. The localizable-text package also supports substitution into thestrings, which can include variables as placeholders. For example, anapplication that needs to report that an operation on a specified account wassuccessful must provide a string like "The operation on accountnumber was successful"; the variable number is to bereplaced by the actual account number. Without support for creatingstrings with variable pieces, each possible string would need its own key, orthe strings would have to be built phrase by phrase.

Both of these approaches quickly become intractable if a variable can takemany values or if a string has several variable components. Instead,the localizable text package supports substitution of variables in stringswith optional arguments. A string in a message catalog uses integers inbraces--for example, {0} or {1}--to represent variablecomponents. Figure 112 shows an example from an English message catalogfor a string with a single variable substitution. (The same key inmessage catalogs for other languages has a translation of this string with thevariable in the appropriate location for the language.)

**Figure 112. A message-catalog entry with a variable substring**

```
successfulTransaction = The operation on account {0} was successful.
```

The values that are substituted into the string come from an array ofoptional arguments. One of the constructors forLocalizableTextFormatter objects takes an array of objects as an argument, andsuch an array of objects can be set within any LocalizableTextFormatterobject. The array is used to hold values for variable parts of astring. When a format method is called on the object, the array ispassed to the format method, which takes an element of the array andsubstitutes it into a placeholder with the matching index in thestring. The value at index 0 in the array replaces the {0} variable inthe string, the value at index 1 replaces {1}, and so forth.

Figure 113 shows the creation of a single-element argument array andthe creation and use of a LocalizableTextFormatter. The element in theargument array is the account number entered by the user. TheLocalizableTextFormatter is created by using a constructor that takes thearray of optional arguments; this can also be set directly by using thesetArguments method on the LocalizableTextFormatter object. Later inthe code, the application calls the format method. The format methodautomatically substitutes values from the array of arguments into the stringreturned from the appropriate message catalog.

**Figure 113. Code example: Formatting a message with a variable substring**

```
public void updateAccount(String transactionType) {  ...  Object[] arg = { new
String(this.accountNumber)};  ...  LocalizableTextFormatter successLTF =      new
LocalizableTextFormatter("BankingResources",
"successfulTransaction",                                "BankingSample",
arg);   ...  successLTF.format(this.applicationLocale);   ...}
```

Nesting LocalizableTextFormatter objectsThe ability to substitute variables into the strings in message catalogs addsa level of flexibility to the localizable-text package, but the additionalflexibility is limited, at least in an international environment, unless thesubstituted arguments themselves can be localized. For example, if anapplication needs to report that an operation on a specific account wassuccessful, a string like "The operation on account number wassuccessful"--where the only variable is an account number--can betranslated and used in message catalogs for multiple languages. Astring in which a variable is also a string, for example, "The typeoperation on account number was successful"--where the newtype variable takes values like "deposit" and"withdrawal"--cannot be as easily translated. The values assumedby the type variable also need to be localized.

Figure 114 shows a message string in an English catalog with twovariables, one of which will be localized, and the keys for two possiblevalues. (The second variable in the string, the account number, issimply a number that must be substituted into the string; it does notneed to be localized.)

**Figure 114. A message-catalog entry with two variable substrings**

```
sucessfulTransaction = The {0} operation on account {1} was successful.depositOpString =
depositwithdrawOpString = withdrawal
```

To support localization of substrings, the localizable-text package allowsthe nesting of LocalizableTextFormatter objects. This is done simply byinserting a LocalizableTextFormatter object into the array of arguments foranother LocalizableTextFormatter. When the format method does thevariable substitution, it formats any LocalizableTextFormatter objects as itsubstitutes array elements for variables. This allows substrings to beformatted independently of the string in which they are embedded.

Figure 115 modifies the example in Figure 113 to format a message with a localizablesubstring. First, a LocalizableTextFormatter object for the localizablesubstring (referring to a deposit operation) is created. This object isinserted, along with the account-number information, into the array ofarguments. The array of arguments is then used in constructing theLocalizableTextFormatter object for the complete string; when the formatmethod is called, the embedded LocalizableTextFormatter object is formatted toreplace the first variable, and the account number is substituted for thesecond variable.

**Figure 115. Code example: Formatting a message with a localizable variable substring**

```
public void updateAccount(String transactionType) {   ...   // Successful Deposit.
LocalizableTextFormatter opLTF =      new LocalizableTextFormatter("BankingResources,
"depositOpString", "BankingSample");   Object[] args = {opLTF, new String(this.accountNumber)};
LocalizableTextFormatter successLTF =       new LocalizableTextFormatter("BankingResources",
"successfulTransaction",                                    "BankingSample",
args);   ...   successLTF.format(this.applicationLocale);   ...}
```

#### Customizing the behavior of a format method

The array of optional arguments can contain simple values, like an accountnumber to be substituted into a formatted string, and otherLocalizableTextFormatter objects, representing localizable substrings to besubstituted into a larger formatted string. These techniques aredescribed in Assembling complex strings. In addition, the optional-argument array can containobjects of user-defined classes.

User-defined classes used as optional arguments provideapplication-specific format methods, which programmers can use to performlocalization on the basis of any number of values, not just locale and timezone. These user-defined classes need to be available only on thesystems where they are constructed and inserted into LocalizableTextFormatterobjects and where the actual formatting is done; client applications donot need to install these classes.

The localizable-text package provides an example of such a user-definedclass in the LocalizableTextDateTimeArgument class. This class allowsdate and time information to be selectively formatted according to the stylevalues defined in the java.text.DateFormat class and accordingto the constants defined by the LocalizableTextDateTimeArgument class.

The DateFormat styles determine how information is reported about adate. For example, when the DateFormat.FULL style is chosen, thetwenty-second day of February in 2000 is represented in English as*Tuesday, February 22, 2000*. When the DateFormat.SHORTstyle is used, the same date is represented as *2/22/00*. Thevalid values are:

- DateFormat.FULL
- DateFormat.LONG
- DateFormat.MEDIUM
- DateFormat.SHORT
- DateFormat.DEFAULT

The LocalizableTextDateTimeArgument class defines constants that can beused to request only date or time information, or both, either in date-timeorder or in time-date order. The defined values are:

- LocalizableTextDateTimeArgument.TIME
- LocalizableTextDateTimeArgument.DATE
- LocalizableTextDateTimeArgument.TIMEANDDATE
- LocalizableTextDateTimeArgument.DATEANDTIME

An object of a user-defined class like the LocalizableTextDateTimeArgumentclass can be set in the optional-argument array of a LocalizableTextFormatterobject, and when the LocalizableTextFormatter object attempts to format theuser-defined object, it calls the format method on that object. Thatformat method, written by the application developer, can do whatever isappropriate with the application-specific values. In the case of theLocalizableTextDateTimeArgument class, the format method determines if date,time, or both are required, formats them according to the DateFormat value,and assembles them in the order requested in theLocalizableTextDateTimeArgument style. The date and time informationare also affected by the locale and time-zone values, but the refinements inthe formatting are accomplished by the DateFormat class and the user-definedvalues.

The string assembled from a user-defined class like theLocalizableTextDateTimeArgument class can then be substituted into a largerstring, just as the return values of nested LocalizableTextFormatter objectscan be. When writing such user-defined classes, it is helpful to thinkof them as specialized versions of the generic LocalizableTextFormatter class,and the way in which the LocalizableTextFormatter class is written provides amodel for writing user-defined classes.

Structure of the LocalizableTextFormatter classThe LocalizableTextFormatter class is a general-purpose class for localizabletext. It extends the java.lang.Object class andimplements the java.io.Serializable interface and fourlocalizable-text interfaces:

- LocalizableTextLTZ
- LocalizableTextL
- LocalizableTextTZ
- LocalizableText

Each of the localizable-text interfaces implemented by theLocalizableTextFormatter class implements the Localizable interface (whichsimply extends the Serializable interface) and defines a single formatmethod:

- The LocalizableTextLTZ interface defines format(locale, timezone).
- The LocalizableTextL defines format(locale).
- The LocalizableTextTZ defines format(timezone).
- The LocalizableText defines format().

Because the LocalizableTextFormatter class implements all four of theseinterfaces, it must provide an implementation for each of these formatmethods.

Writing a user-defined classA user-defined class must implement at least one of the localizable-textinterfaces and its corresponding format method, as well as the Serializableinterface. If the class implements more than one of thelocalizable-text interfaces and format methods, the order of evaluation of theinterfaces is:

1. LocalizableTextLTZ
2. LocalizableTextL
3. LocalizableTextTZ
4. LocalizableText

For example, the LocalizableTextDateTimeArgument class implements only theLocalizableTextLTZ interface, as shown in Figure 116.

**Figure 116. Code example: The structure of the LocalizableTextDateTimeArgument class**

```
package com.ibm.websphere.i18n.localizabletext;import java.util.Locale;import java.util.Date;import
java.text.DateFormat;import java.util.TimeZone;import java.io.Serializable;public class
LocalizableTextDateTimeArgument implements LocalizableTextLTZ,
Serializable{   ...}
```

A user-defined class must contain a constructor and an implementation ofthe format methods as defined in the localizable-text interfaces that theclass implements. It can also contain other methods as needed.The LocalizableTextDateTimeArgument class contains a constructor, a singleformat method, an equality method, a hash-code generator, and astring-conversion method.

**Figure 117. Code example: The methods in the LocalizableTextDateTimeArgument class**

```
...public class LocalizableTextDateTimeArgument implements LocalizableTextLTZ,
Serializable{   public final static int DATE = 1;   public final static int TIME = 2;   public final
static int DATEANDTIME = 3;   public final static int TIMEANDDATE = 4;   private Date date = null;
private dateTimeStyle = LocalizableTextDateTimeArgument.DATE;   private int dateFormatStyle =
DateFormat.FULL;   ...   public LocalizableTextDateTimeArgument(Date date, int dateTimeStyle,
int dateFormatStyle)   { ... }   public boolean equals(Object param)   { ... }   public format
(Locale locale, TimeZone timeZone)      throws IllegalArgumentException   { ... }    public int
hashCode()   { ... }    public String toString()   { ... }}
```

Each format method in the user-defined class can do whatever is appropriatefor the application. In the LocalizableTextDateTimeArgument class, theformat method (see Figure 118 for the implementation) examines the setting of thedate-time style set within the object, for example, DATEANDTIME. Itthen assembles the requested information in the requested order, according tothe date-format value.

**Figure 118. Code example: The format method in the LocalizableTextDateTimeArgument class**

```
public format (Locale locale, TimeZone timeZone)   throws IllegalArgumentException{   String
returnString = null;      switch(dateTimeStyle) {      case LocalizableTextDateTimeArgument.DATE :
{         returnString = DateFormat.getDateInstance(dateFormatStyle,
locale).format(date);         break;      }      case LocalizableTextDateTimeArgument.TIME :      {
df = DateFormat.getTimeInstance(dateFormatStyle, locale);         df.setTimeZone(timeZone);
returnString = df.format(date);         break;      }      case
LocalizableTextDateTimeArgument.DATEANDTIME :      {         dateString =
DateFormat.getDateInstance(dateFormatStyle,
locale).format(date);         df = DateFormat.getTimeInstance(dateFormatStyle, locale);
df.setTimeZone(timeZone);         timeString = df.format(date);         returnString = dateString +
" " + timeString;         break;      }      case LocalizableTextDateTimeArgument.TIMEANDDATE :
{         dateString = DateFormat.getDateInstance(dateFormatStyle,
locale).format(date);         df = DateFormat.getTimeInstance(dateFormatStyle, locale);
df.setTimeZone(timeZone);         returnString = timeString + " " + dateString;         break;
}      default :      {         throw new IllegalArgumentException();      }   }   return
returnString;}
```

An application can create a LocalizableTextDateTimeArgument object (or anobject of any other user-defined class) and place it in the optional-argumentarray of a LocalizableTextFormatter object. When theLocalizableTextFormatter object reaches the user-defined object, it willattempt to format it by calling the object's format method. Thereturned string is then substituted for a variable as theLocalizableTextFormatter processes each element in the array of optionalarguments.

## Deploying the formatter enterprise bean

The localizable-text package provides a stateless session enterprise bean, theLocalizableTextResourceAccessorBean, for formatting messages in a distributedenvironment. The format methods on a LocalizableTextFormatter objecttransparently look up and contact the session bean. However, thesession bean must be deployed into a WebSphere Application Server before itcan be used. When an application calls a format method on aLocalizableTextFormatter object, the format method uses the name of theformatting application to find a server where a formatting bean has beendeployed. This localizable-text bean assembles a string frominformation in the LocalizableTextFormatter object and returns the assembledstring.

The localizable-text package provides a command-line Java tool,LocalizableTextEJBDeploy, for deploying the localizable-text session bean, andthe package provides all the code necessary to run the session bean. Anadministrator uses the tool to deploy and name the formatting bean. Thename given to the bean must match the name specified inLocalizableTextFormatter objects as the name of the formattingapplication. The tool can also be used to remove deployed beans whenthey are no longer needed.

#### Setting up the tool

Before the LocalizableTextEJBDeploy tool can be used to deploy a formattingsession bean for localizable applications, the following conditions must bemet:

- A directory called temp must exist under the WebSphere installationdirectory. This is typically created during the installation ofWebSphere Application Server, and if it does not exist, it must becreated.
- The file ujc.jar must be present on the CLASSPATH variable.This file contains the compiled Java code for the deployment tool.

#### Deploying a formatting session bean

After the prerequisites for the tool have been met, the tool can be used todeploy formatting session beans. The tool requires values for fourarguments and has two optional arguments:

```
LocalizableTextEJBDeploy -a <appName> -h <hostName>                          -i <installationDir> -x
<action>                        [ -s <serverName> ] [ -c <containerName> ]
```

The required arguments, which can be specified in any order, follow:

- appName: The name of the formatting session bean. This nameis used in LocalizableTextFormatter objects to specify where the actualformatting takes place. If a LocalizableTextFormatter object specifiesa name that cannot be resolved, an exception is thrown by the formatmethod.
- hostName: The name of the machine on which the formatting sessionbean is deployed. This value specified here is case sensitive on allplatforms.
- installationDir: The location at which WebSphere Application Serveris installed on the machine.
- action: The task that the tool is being used to perform. Thetool is used to create the deployment information for formatting session beansand to remove the deployment information when the beans are no longerneeded. There are two possible values for this argument:
  - create: The tool creates the following JAR and XML files for theformatter session bean and deletes them when deployment is complete:
    - <installRoot>/temp/LocalizableText-Jetace-<appName>.xml
    - <installRoot>/temp/LocalizableText-XMLConfig-<appName>.xml
    - <installRoot>/deployableEJBs/LocalizableText-<appName>.jar
    - <installRoot>/deployedEJBs/DeployedLocalizableText-<appName>.jar
  - delete: The tool creates the following XML file for the formattersession bean:
    - <installRoot>/temp/LocalizableText-XMLConfig-<appName>.xml

The optional arguments, which can also be specified in any order,follow:

- serverName: The name of the WebSphere Application Server. Ifthis argument is not specified, the value "Default Server" is used.
- containerName: The name of the container within WebSphereApplication Server. If this argument is not specified, the value"Default Container" is used.

The formatting bean can be deployed on multiple systems, as long as eachsystem has a copy of the necessary resource bundles. Figure 119 illustrates the commands for deploying a formatting beancalled CheckingApplication on two machines, a UNIX machine calledResourcesHost1 and a PC called ResourcesHost2.

**Figure 119. Deploying a formatting enterprise bean**

```
% java LocalizableTextEJBDeploy -a CheckingApplication -x create-h ResourcesHost1 -i
/usr/WebSphere/AppServerC:\java LocalizableTextEJBDeploy -a CheckingApplication -x create-h
ResourcesHost2-i C:\WebSphere\AppServer
```

When the formatting bean is no longer needed, it can be deleted with theLocalizableTextEJBDeploy tool. Figure 120 shows the command for removing the formatting bean deployedin Figure 119 from one of the machines.

**Figure 120. Deleting a deployed formatting enterprise bean**

```
C:\java LocalizableTextEJBDeploy -a CheckingApplication -x delete-h ResourcesHost2-i
C:\WebSphere\AppServer
```

# Developing enterprise beans

This chapter explains the basic tasks required to develop and package the mostcommon types of enterprise beans. Specifically, this chapter focuses oncreating stateless session beans and entity beans that use container-managedpersistence (CMP); in the discussion of stateless session beans,important information about stateful beans is also provided. Forinformation on developing entity beans that use bean-managed persistence(BMP), see Developing entity beans with BMP.

The information in this chapter is not exhaustive; however, itincludes the information you need to develop basic enterprise beans.For information on developing more complicated enterprise beans, consult acommercially available book on enterprise bean development. The exampleenterprise beans discussed in this chapter and the example Java applicationsand servlets that use them are described in Information about the examples described in the documentation.

This chapter describes the requirements for building each of the majorcomponents of an enterprise bean. If you do *not* intend touse one of the commercially available integrated development environments(IDE), such as IBM's VisualAge for Java, you must build each of thesecomponents manually (by using tools in the Java Development Kit andWebSphere). Manually developing enterprise beans is much more difficultand error-prone than developing them in an IDE. Therefore, it isstrongly recommended that you choose an IDE with which you arecomfortable.

**Note:**

In the EJB server (CB) environment, do not duplicate unqualified interfaceand exception names in enterprise beans. For example, thecom.ibm.ejs.doc.account.Account interfacemust not be reused in a package namedcom.ibm.ejs.doc.bank.Account. Thisrestriction is necessary because the EJB server (CB) tools generate enterprisebean support files that use the unqualified name only.

---

## Developing entity beans with CMP

In an entity bean with CMP, the container handles the interactions between theentity bean and the data source. In an entity bean with BMP, the entitybean must contain all of the code required for the interactions between theentity bean and the data source. For this reason, developing an entitybean with CMP is simpler than developing an entity bean with BMP.

This section examines the development of entity beans with CMP.While much of the information in this section also applies to entity beanswith BMP, there are some major differences between the two types. Forinformation on the tasks required to develop an entity bean with BMP, see Developing entity beans with BMP.

Every entity bean must contain the following basic parts:

- The enterprise bean class. For more information, see Writing the enterprise bean class (entity with CMP).
- The enterprise bean's home interface. For more information,see Writing the home interface (entity with CMP).
- The enterprise bean's remote interface. For more information,see Writing the remote interface (entity with CMP).
- The enterprise bean's primary key class. For more information,see Writing the primary key class (entity with CMP).

### Writing the enterprise bean class (entity with CMP)

In a CMP entity bean, the bean class defines and implements the businessmethods of the enterprise bean, defines and implements the methods used tocreate instances of the enterprise bean, and implements the methods used bythe container to inform the instances of the enterprise bean of significantevents in the instance's life cycle. Enterprise bean clients neveraccess the bean class directly; instead, the classes that implement thehome and remote interfaces are used to indirectly invoke the methods definedin the bean class.

By convention, the enterprise bean class is named *Name*Bean,where *Name* is the name you assign to the enterprise bean.The enterprise bean class for the example Account enterprise bean is namedAccountBean.Every entity bean class with CMP must meet the following requirements:

- It must be public, it must *not* be abstract, and it mustimplement the javax.ejb.EntityBean interface. For moreinformation, see Implementing the EntityBean interface.
- It must define instance variables that correspond to persistent dataassociated with the enterprise bean. For more information, see Defining variables.
- It must implement the business methods used to access and manipulate thedata associated with the enterprise bean. For more information, see Implementing the business methods.
- It must define and implement an ejbCreate method for each way in which theenterprise bean can be instantiated. A corresponding ejbPostCreatemethod must be defined for each ejbCreate method. For more information,see Implementing the ejbCreate and ejbPostCreate methods.

**Note:**

The enterprise bean class can implement the enterprise bean's remoteinterface, but this is not recommended. If the enterprise bean classimplements the remote interface, it is possible to inadvertently pass the*this* variable as a method argument.

An enterprise bean class cannot implement two different interfaces if themethods in the interfaces have the same name, even if the methods havedifferent signatures, due to the Java-IDL mapping specification. Errorscan occur when the enterprise bean is deployed.

Figure 18 shows the main parts of the enterprise bean class for theexample Account enterprise bean. (Emphasized code is in boldtype.) The sections that follow discuss these parts in greaterdetail.

**Figure 18. Code example: The AccountBean class**

```
...import java.util.Properties;import javax.ejb.*;import java.lang.*;public class AccountBean
implements EntityBean {      // Set instance variables here     ...     // Implement methods here
...}
```

#### Defining variables

An entity bean class can contain both persistent and nonpersistent instancevariables; however, static variables are not supported in enterprisebeans unless they are also final (that is, they are constants). Staticvariables are not supported because there is no way to guarantee that theyremain consistent across enterprise bean instances.

Container-managed fields (which are persistent variables) are stored in adatabase. Container-managed fields must be public.

Nonpersistent variables are *not* stored in a database and aretemporary. Nonpersistent variables must be used with caution and mustnot be used to maintain the state of an EJB client between methodinvocations. This restriction is necessary because nonpersistentvariables cannot be relied on to remain the same between method invocationsoutside of a transaction because other EJB clients can change these variables,or they can be lost when the entity bean is passivated.

**Note:**

> In the EJB server (CB) environment, container-managed fields in entity beansmust be valid for use in CORBA IDL files. Specifically, the variablenames must use ISO Latin-1 characters, they must *not* begin with anunderscore character (_), they must *not* contain the dollarcharacter ($), and they must *not* be CORBA keywords.Variables that have the same name but different cases are not allowed.(For example, you cannot use the following variables in the same class:*accountId* and *AccountId*. For more information onCORBA IDL, consult a CORBA programming guide.
>
> Also, container-managed fields in entity beans must be valid Java types,but they *cannot* be of type javax.ejb.Handle or anarray of type EJBObject or EJBHome.

The AccountBean class contains three container-managed fields (shown in Figure 19):

- *accountId*, which identifies the account ID associated with anaccount
- *type*, which identifies the account type as either savings (1)or checking (2)
- *balance*, which identifies the current balance of the account

**Figure 19. Code example: The variables of the AccountBean class**

```
...public class AccountBean implements EntityBean {      private EntityContext entityContext = null;
private ListResourceBundle bundle =           ResourceBundle.getBundle(
"com.ibm.ejs.doc.account.AccountResourceBundle");     public long accountId = 0;     public int type
= 1;     public float balance = 0.0f;     ...}
```

The deployment descriptor is used to identify container-managed fields inentity beans with CMP. In an entity bean with CMP, eachcontainer-managed field must be initialized by each ejbCreate method (see Implementing the ejbCreate and ejbPostCreate methods).

A subset of the container-managed fields is used to define the primary keyclass associated with each instance of an enterprise bean. As is shownin Writing the primary key class (entity with CMP), the *accountId* variable defines the primary keyfor the Account enterprise bean.The AccountBean class contains two nonpersistent variables:

- *entityContext*, which identifies the entity context of eachinstance of an Account enterprise bean. The entity context can be usedto get a reference to the EJB object currently associated with the beaninstance and to get the primary key object associated with that EJBobject.
- *bundle*, which encapsulates a resource bundle class(com.ibm.ejs.doc.account.AccountResourceBundle)that contains locale-specific objects used by the Account bean.

#### Implementing the business methods

The business methods of an entity bean class define the ways in which the dataencapsulated in the class can be manipulated. The business methodsimplemented in the enterprise bean class cannot be directly invoked by an EJBclient. Instead, the EJB client invokes the corresponding methodsdefined in the enterprise bean's remote interface, by using an EJB objectassociated with an instance of the enterprise bean, and the container invokesthe corresponding methods in the instance of the enterprise bean.

Therefore, for every business method implemented in the enterprise beanclass, a corresponding method must be defined in the enterprise bean'sremote interface. The enterprise bean's remote interface isimplemented by the container in the EJB object class when the enterprise beanis deployed.

Figure 20 shows the business methods for the AccountBean class.These methods are used to add a specified amount to an account balance andreturn the new balance (add), to return the current balance of an account(getBalance), to set the balance of an account (setBalance), and to subtract aspecified amount from an account balance and return the new balance(subtract).The subtract method throws the user-defined exceptioncom.ibm.ejs.doc.account.InsufficientFundsExceptionif a client attempts to subtract more money from an account than is containedin the account balance. The subtract method in the Account bean'sremote interface must also throw this exception as shown in Figure 25. User-defined exception classes for enterprise beansare created as are any other user-defined exception class. The messagecontent for the InsufficientFundsException exception is obtained from theAccountResourceBundle class file by invoking the getMessage method on the*bundle* object.

**Note:**

If an enterprise bean container catches a system exception from the businessmethod of an enterprise bean, and the method is running within acontainer-managed transaction, the container rolls back the transaction beforepassing the exception on to the client. However, if the business methodis throwing an application exception, then the transaction is not rolled back(it is committed), unless the application has called setRollbackOnlyfunction. In this case, the transaction is rolled back before theexception is re-thrown.

**Note:**

In the EJB server (CB) environment, use of underscores (_) in the names ofuser-defined interfaces and exception classes is discouraged.

**Figure 20. Code example: The business methods of the AccountBean class**

```
...public class AccountBean implements EntityBean {      ...     public long accountId = 0;
public int type = 1;     public float balance = 0.0f;     ...     public float add(float amount) {
balance += amount;          return balance;     }     ...     public float getBalance() {
return balance;      }     ...     public void setBalance(float amount) {           balance =
amount;      }     ...     public float subtract(float amount) throws InsufficientFundsException {
if(balance < amount) {               throw new InsufficientFundsException(
bundle.getMessage("insufficientFunds"));          }          balance -= amount;          return
balance;     }     ...}
```

#### Standard application exceptions for entity beans

Version 1.1 of the EJB specification defines several standardapplication exceptions for use by enterprise beans. All of theseexceptions are subclasses of the javax.ejb.EJBExceptionclass. For entity beans with both container- and bean-managedpersistence, the EJB specification defines the following applicationexceptions:

- javax.ejb.CreateException
- javax.ejb.DuplicateKeyException
- javax.ejb.RemoveException
- javax.ejb.FinderException
- javax.ejb.ObjectNotFoundException

Application programmers can use the generic EJBException class or one ofthe provided subclassed exceptions, or programmers can define their ownexceptions by subclassing any of this family of exceptions. All ofthese exceptions inherit from the javax.ejb.RuntimeExceptionclass and do not have to be explicitly declared in throws clauses.

Each exception is discussed in more detail within the relevantsection; for more information on:

- CreateException and DuplicateKeyException (a subclass of theCreateException class), see Implementing the ejbCreate and ejbPostCreate methods.
- javax.ejb.RemoveException, see Implementing the EntityBean interface.
- FinderException and ObjectNotFoundException (a subclass of theFinderException class), see Defining finder methods.

**Note:**

Version 1.0 of the EJB specification used thejava.rmi.RemoteException class to capture application-specificexceptions; the EJBException class and its subclasses are new in the1.1 version of the specification. Therefore, using theRemoteException class is now deprecated in favor of the more precise exceptionclasses. Older applications that use the RemoteException class canstill run, but enterprise beans compliant with version 1.1 of thespecification must use the new exception classes.

#### Implementing the ejbCreate and ejbPostCreate methods

You must define and implement an ejbCreate method for each way in which youwant a new instance of an enterprise bean to be created. For eachejbCreate method, you must also define a corresponding ejbPostCreatemethod. Each ejbCreate and ejbPostCreate method must correspond to acreate method in the home interface.

Like the business methods of the bean class, the ejbCreate andejbPostCreate methods cannot be invoked directly by the client.Instead, the client invokes the create method of the enterprise bean'shome interface by using the EJB home object, and the container invokes theejbCreate method followed by the ejbPostCreate method. If the ejbCreateand ejbPostCreate methods are executed successfully, an EJB object is createdand the persistent data associated with that object is inserted into the datasource.

For an entity bean with CMP, the container handles the required interactionbetween the entity bean instance and the data source between calls to theejbCreate and ejbPostCreate methods. For an entity bean with BMP, theejbCreate method must contain the code to directly handle thisinteraction. For more information on entity beans with BMP, see Developing entity beans with BMP.

Each ejbCreate method in an entity bean with CMP must meet the followingrequirements:

- It must be public and return the same type as the primary key. Theactual return value must be null.
- Its arguments must be valid for Java remote method invocation(RMI). For more information, see The java.io.Serializable and java.rmi.Remote interfaces.
- It must initialize the container-managed fields of the enterprise beaninstance. The container extracts the values of these variables andwrites them to the data source after the ejbCreate method returns.

Each ejbPostCreate method must be public, return void, and have the samearguments as the matching ejbCreate method.If necessary, both the ejbCreate method and the ejbPostCreate method can throwthe javax.ejb.EJBException exception or one of thecreation-related subclasses, the CreateException or the DuplicateKeyExceptionexceptions. The DuplicateKeyException class is a subclass of theCreateException class. Throwing thejava.rmi.RemoteException exception is deprecated; see Standard application exceptions for entity beans for more information.

Figure 21 shows two sets of ejbCreate and ejbPostCreate methodsrequired for the example AccountBean class. The first set of ejbCreateand ejbPostCreate methods are wrappers that call the second set of methods andset the *type* variable to 1 (corresponding to a savings account) andthe *balance* variable to 0 (zero dollars).

**Figure 21. Code example: The ejbCreate and ejbPostCreate methods of the AccountBean class**

```
...public class AccountBean implements EntityBean {      ...     public long accountId = 0;
public int type = 1;     public float balance = 0.0f;     ...     public Integer
ejbCreate(AccountKey key) {          ejbCreate(key, 1, 0.0f);     }     ...     public Integer
ejbCreate(AccountKey key, int type, float initialBalance)      throws EJBException {
accountId = key.accountId;          type = type;          balance = initialBalance;     }     ...
public void ejbPostCreate(AccountKey key)      throws EJBException {          ejbPostCreate(key, 1,
0);     }     ...     public void ejbPostCreate(AccountKey key, int type, float initialBalance) { }
...}
```

#### Implementing the EntityBean interface

Each entity bean class must implement the methods inherited from thejavax.ejb.EntityBean interface. The container invokesthese methods to inform the bean instance of significant events in theinstance's life cycle. (For more information, see Entity bean life cycle.) All of these methods must be public and returnvoid; they can throw the javax.ejb.EJBException exceptionor, in the case of the ejbRemove method, thejavax.ejb.RemoveException exception. Throwing thejava.rmi.RemoteException exception is deprecated; see Standard application exceptions for entity beans for more information.

- ejbActivate--This method is invoked by the container when thecontainer selects an entity bean instance from the instance pool and assignsthat instance to a specific existing EJB object. This method mustcontain any code that you want to execute when the enterprise bean instance isactivated.
- ejbLoad--This method is invoked by the container to synchronize anentity bean's container-managed fields with the corresponding data in thedata source. (That is, the values of the fields in the data source areloaded into the container-managed fields in the corresponding enterprise beaninstance.) This method must contain any code that you want to executewhen the enterprise bean instance is synchronized with associated data in thedata source.
- ejbPassivate--This method is invoked by the container when thecontainer disassociates an entity bean instance from its EJB object and placesthe enterprise bean instance in the instance pool. This method mustcontain any code that you want to execute when the enterprise bean instance is"passivated" or deactivated.
- ejbRemove--This method is invoked by the container when a clientinvokes the remove method inherited by the enterprise bean's homeinterface from the javax.ejb.EJBHome interface. Thismethod must contain any code that you want to execute before an enterprisebean instance is removed from the container (and the associated data isremoved from the data source). This method can throw thejavax.ejb.RemoveException exception if removal of an enterprisebean instance is not permitted.
- setEntityContext--This method is invoked by the container to pass areference to the javax.ejb.EntityContext interface to anenterprise bean instance. If an enterprise bean instance needs to usethis context at any time during its life cycle, the enterprise bean class mustcontain an instance variable to store this value. This method mustcontain any code required to store a reference to a context.
- ejbStore--This method is invoked by the container when the containerneeds to synchronize the data in the data source with the values of thecontainer-managed fields in an enterprise bean instance. (That is, thevalues of the variables in the enterprise bean instance are copied to the datasource, overwriting the previous values.) This method must contain anycode that you want to execute when the data in the data source is overwrittenwith the corresponding values in the enterprise bean instance.
- unsetEntityContext--This method is invoked by the container, beforean enterprise bean instance is removed, to free up any resources associatedwith the enterprise bean instance. This is the last method called priorto removing an enterprise bean instance.

In entity beans with CMP, the container handles the required data sourceinteraction for these methods. In entity beans with BMP, these methodsmust directly handle the required data source interaction. For moreinformation on entity beans with BMP, see More-advanced programming concepts for enterprise beans.

These methods have several possible uses, including the following:

- They can contain audit or debugging code.
- They can contain code for allocating and deallocating additional resourcesused by the bean instance (for example, an SNA connection to amainframe).

As shown in Figure 22, except for the setEntityContext and unsetEntityContextmethods, all of these methods are empty in the AccountBean class because noadditional action is required by the bean for the particular life cycle statesassociated with the these methods. The setEntityContext andunsetEntityContext methods are used in a conventional way to set the value ofthe *entityContext* variable.

**Figure 22. Code example: Implementing the EntityBean interface in the AccountBean class**

```
...public class AccountBean implements EntityBean {      private EntityContext entityContext = null;
...     public void ejbActivate() throws EJBException { }     ...     public void ejbLoad () throws
EJBException { }     ...     public void ejbPassivate() throws EJBException { }     ...     public
void ejbRemove() throws EJBException { }     ...     public void ejbStore () throws EJBException { }
...     public void setEntityContext(EntityContext ctx) throws EJBException {          entityContext
= ctx;      }     ...     public void unsetEntityContext() throws EJBException {
entityContext = null;      }}
```

## Writing the home interface (entity with CMP)

An entity bean's home interface defines the methods used by clients tocreate new instances of the bean, find and remove existing instances, andobtain metadata about an instance. The home interface is defined by theenterprise bean developer and implemented in the EJB home class created by thecontainer during enterprise bean deployment.

The container makes the home interface accessible to enterprise beanclients through the Java Naming and Directory Interface (JNDI). JNDI isindependent of any specific naming and directory service and allows Java-basedapplications to access any naming and directory service in a standardway.

By convention, the home interface is named *Name*Home, where*Name* is the name you assign to the enterprise bean. Forexample, the Account enterprise bean's home interface is namedAccountHome.Every home interface must meet the following requirements:

- It must extend the javax.ejb.EJBHome interface. Thehome interface inherits several methods from thejavax.ejb.EJBHome interface. See The javax.ejb.EJBHome interface for information on thesemethods.
- Each method in the interface must be either a create method thatcorresponds to a set of ejbCreate and ejbPostCreate methods in the EJB objectclass, or a finder method. For more information, see Defining create methods and Defining finder methods.
- The parameters and return value of each method defined in the homeinterface must be valid for Java RMI. For more information, see The java.io.Serializable and java.rmi.Remote interfaces. In addition, each method's throws clause mustinclude the java.rmi.RemoteException exception class.

Figure 23 shows the relevant parts of the definition of the homeinterface (AccountHome) for the example Account bean. This interfacedefines two abstract create methods: the first creates an Account objectby using an associated AccountKey object, the second creates an Account objectby using an associated AccountKey object and specifying an account type and aninitial balance. The interface defines the required findByPrimaryKeymethod and a findLargeAccounts method, which returns a collection of accountscontaining balances greater than a specified amount.

**Figure 23. Code example: The AccountHome home interface**

```
...import java.rmi.*;import java.util.*;import javax.ejb.*;public interface  AccountHome extends
EJBHome {     ...     Account create (AccountKey id) throws CreateException, RemoteException;
...     Account create(AccountKey id, int type, float initialBalance)          throws
CreateException, RemoteException;     ...     Account findByPrimaryKey (AccountKey id)
RemoteException, FinderException;     ...     Enumeration findLargeAccounts(float amount)
throws RemoteException, FinderException;}
```

#### Defining create methods

A create method is used by a client to create an enterprise bean instance andinsert the data associated with that instance into the data source.Each create method must be named create and it must have the same number andtypes of arguments as a corresponding ejbCreate method in the enterprise beanclass. (The ejbCreate method must itself have a correspondingejbPostCreate method.)

Each create method must meet the following requirements:

- It must be named create.
- It must return the type of the enterprise bean's remoteinterface. For example, the return type for the create methods in theAccountHome interface is Account (as shown in Figure 23).
- It must have a throws clause that includes thejava.rmi.RemoteException exception, thejavax.ejb.CreateException exception, and all of the applicationexceptions defined in the throws clause of the corresponding ejbCreate andejbPostCreate methods.

#### Defining finder methods

A finder method is used to find one or more existing entity EJBobjects. Each finder method must be named find*Name*, where*Name* further describes the finder method's purpose.

At minimum, each home interface must define the findByPrimaryKey methodthat enables a client to locate an EJB object by using the primary keyonly. The findByPrimaryKey method has one argument, an object of thebean's primary key class, and returns the type of the bean's remoteinterface.

Every other finder method must meet the following requirements:

- It must return the type of the enterprise bean's remote interface,the java.util.Enumeration interface, or thejava.util.Collection interface (when a finder method can returnmore than one EJB object or an EJB collection).
- It must have a throws clause that includes thejava.rmi.RemoteException andjavax.ejb.FinderException exception classes.

While every entity bean must contain the default finder method, you canwrite additional finder methods if needed. For example, the Accountbean's home interface defines the findLargeAccounts method to findobjects that encapsulate accounts with balances of more than a specifiedamount, as shown in Figure 24. Because this finder method can be expected to returna reference to more than one EJB object, its return type isEnumeration.

**Figure 24. Code example: The findLargeAccounts method**

```
Enumeration findLargeAccounts(float amount)            throws RemoteException, FinderException;
```

Every EJB server can implement the findByPrimaryKey method. Duringenterprise bean deployment, the container generates the code required tosearch the database for the appropriate enterprise bean instance.

However, for each additional finder method that you define in the homeinterface, the enterprise bean deployer must associate finder logic with thatfinder method. This logic is used by the EJB server during deploymentto generate the code required to implement the finder method.

The EJB Specification does not define the format of the finder logic, sothe format can vary according to the EJB server you are using. For moreinformation on creating finder logic, see Creating finder logic in the EJB server (AE) or Creating finder logic in the EJB server (CB).

## Writing the remote interface (entity with CMP)

An entity bean's remote interface provides access to the business methodsavailable in the bean class. It also provides methods to remove an EJBobject associated with a bean instance and to obtain the bean instance'shome interface, object handle, and primary key. The remote interface isdefined by the enterprise bean developer and implemented in the EJB objectclass created by the container during enterprise bean deployment.

By convention, the remote interface is named *Name*, where*Name* is the name you assign to the enterprise bean. Forexample, the Account enterprise bean's remote interface is namedAccount.Every remote interface must meet the following requirements:

- It must extend the javax.ejb.EJBObject interface. Theenterprise bean's remote interface inherits several methods from thejavax.ejb.EJBObject interface. See Methods inherited from javax.ejb.EJBObject for information on these methods.
- You must define a corresponding business method for every business methodimplemented in the enterprise bean class.
- The parameters and return value of each method defined in the interfacemust be valid for Java RMI. For more information, see The java.io.Serializable and java.rmi.Remote interfaces.
- Each method's throws clause must include thejava.rmi.RemoteException exception class.

**Note:**

In the EJB server (CB) environment, do not use method names in the remoteinterface that match method names in the Component Broker Managed ObjectFramework (that is, methods in theIManagedServer::IManagedObjectWithCachedDataObject,CosStream::Streamable, CosLifeCycle::LifeCycleObject,and CosObjectIdentity::IdentifiableObject interfaces). Formore information on the Managed Object Framework, see the Component BrokerProgramming Guide. In addition, do not use underscores (_) atthe end of property or method names; this restriction prevents namecollision with queryable attributes in business object interfaces thatcorrespond to container-managed fields.

Figure 25 shows the relevant parts of the definition of the remoteinterface (Account) for the example Account enterprise bean. Thisinterface defines four methods for displaying and manipulating the accountbalance that exactly match the business methods implemented in the AccountBeanclass.All of the business methods in the remote interface throw thejava.rmi.RemoteException exception class. In addition,the subtract method must throw the user-defined exceptioncom.ibm.ejs.doc.account.InsufficientFundsExceptionbecause the corresponding method in the bean class throws thisexception. Furthermore, any client that calls this method must eitherhandle the exception or pass it on by throwing it.

**Figure 25. Code example: The Account remote interface**

```
...import java.rmi.*;import javax.ejb.*;public interface Account extends EJBObject {     ...
float add(float amount) throws RemoteException;     ...     float getBalance() throws
RemoteException;     ...     void setBalance(float amount) throws RemoteException;     ...     float
subtract(float amount) throws InsufficientFundsException,          RemoteException;}
```

## Writing the primary key class (entity with CMP)

Within a container, every entity EJB object has a unique identity that isdefined by using a combination of the object's home interface name andits primary key, the latter of which is assigned to the object atcreation. If two EJB objects have the same identity, they areconsidered identical.

Primary keys are specified in two ways:

- Simple primary keys, which map to a single field in the entity bean classand are comprised of primitive Java data types (such as integer or long), arespecified in the deployment descriptor.
- Composite primary keys, which map to multiple fields in the entity beanclass (or to data structures built from the primitive Java data types), mustbe encapsulated in a *primary key class*. More complicatedenterprise beans are likely to have composite primary keys, with multipleinstance variables representing the primary key.

The primary key class is used to manage an EJB object's primarykey. By convention, the primary key class is named *Name*Key,where *Name* is the name of the enterprise bean. For example,the Account enterprise bean's primary key class is namedAccountKey.The primary key class must meet the following requirements:

- It must be public and it must be serializable. For moreinformation, see The java.io.Serializable and java.rmi.Remote interfaces.
- Its instance variables must be public, and the variable names must match asubset of the container-managed field names defined in the enterprise beanclass.
- It must have a public default constructor, at a minimum.

**Note:**

For the EJB server (AE) environment, the primary key class of a CMP entitybean must override the equals method and the hashCode method inherited fromthe java.lang.Object class.

Figure 26 shows a composite primary key class for an exampleenterprise bean, Item. In effect, this class acts as a wrapper aroundthe string variables *productId* and *vendorId*. ThehashCode method for the ItemKey class invokes the corresponding hashCodemethod in the java.lang.String class after creating a temporarystring object by using the value of the *productId* variable.In addition to the default constructor, the ItemKey class also defines aconstructor that sets the value of the primary key variables to the specifiedstrings.

**Figure 26. Code example: The ItemKey primary key class**

```
...import java.io.*;// Composite primary key classpublic class ItemKey implements
java.io.Serializable {                                public String productId;      public String
vendorId;      // Constructors     public ItemKey() { };      public ItemKey(String productId,
String vendorId) {           this.productId = productId;           this.vendorId = vendorId;      }
public String getProductId() {           return productId;      }      public String getVendorId() {
return vendorId;       }  ...     // EJB server (AE)-specific method        public boolean
equals(Object other) {              if (other instanceof ItemKey) {                    return
(productId.equals(((ItemKey)                                  other).productId)
&& vendorId.equals(((ItemKey)                                  other).vendorId));                }
else                    return false;        }      ...     // EJB server (AE)-specific method
public int hashCode() {                 return (new productId.hashCode());        }  }
```

A primary key class can also be used to encapsulate a primary key that isnot known ahead of time -- for instance, if the entity bean is intendedto work with several persistent data stores, each of which requires adifferent primary key structure. The entity bean's primary keytype is derived from the primary key type used by the underlying database thatstores the entity objects; it does not necessarily have to be known tothe enterprise bean developer.

To specify an unknown primary key, do the following:

- Declare the argument of the findByPrimaryKey class asjava.lang.Object.
- Declare the return value of the ejbCreate method asjava.lang.Object
- In the deployment descriptor, specify the primary key class as being ofthe type java.lang.Object.

When the primary key selection is deferred to deployment, clientapplications cannot use methods that rely on knowledge of the primary keytype. In addition, applications cannot always depend on methods thatreturn the type of the primary key (such as theEntityContext.getPrimaryKey method) because the return type isdetermined at deployment.

### Interacting with databases

**Note:**

This section applies only to the Advanced Edition EJB environment.Component Broker has its own means of controlling caching; see theComponent Broker Advanced Programming Guide for details.

This section contains general information and tips on enterprise beans anddatabase access.

- Although it is not necessary, it is good practice to specify the user IDand password for a data source either in the enterprise bean to be using thedata source, or in the container of the bean.
- The container supports Option A and Option C caching. When Option Acaching is in use, the application server hosting the enterprise beancontainer must be the only updater of the data in the persistent store.As such, Option A caching is incompatible with the following:
  - Workload managed servers (such as a cluster of clones)
  - Databases with data being shared among multiple applications

  The default caching option is C (multiple entity bean instances, possiblyin different servers, can update bean state in the database). Thedefault caching option can be changed from Option C to Option A by selecting"exclusive persistent store" in the administrative console when creating theentity bean.

Shared database access corresponds to Option C caching. Option A andOption C caching are also known as commit option A and commit option C,respectively.

---

## Developing session beans

In their basic makeup, session beans are similar to entity beans.However, their purposes are very different.

From a component perspective, one of the biggest differences between thetwo types of enterprise beans is that session beans do not have a primary

keyclass and the session bean's home interface does not define findermethods. Session enterprise beans do not require primary keys andfinder methods because session EJB objects are created, associated with aspecific client, and then removed as needed, whereas entity EJB objectsrepresent permanent data in a data source and can be uniquely identified witha primary key. Because the data for session beans is never permanentlystored, the session bean class does not have methods for storing data to andloading data from a data source.

Every session bean must contain the following basic parts:

- The enterprise bean class. For more information, see Writing the enterprise bean class (session).
- The enterprise bean's home interface. For more information,see Writing the home interface (session).
- The enterprise bean's remote interface. For more information,see Writing the remote interface (session).

## Writing the enterprise bean class (session)

A session bean class defines and implements the business methods of theenterprise bean, implements the methods used by the container during thecreation of enterprise bean instances, and implements the methods used by thecontainer to inform the enterprise bean instance of significant events in theinstance's life cycle. By convention, the enterprise bean class isnamed *Name*Bean, where *Name* is the name you assign to theenterprise bean. The enterprise bean class for the example Transferenterprise bean is named TransferBean.Every session bean class must meet the following requirements:

- It must define and implement the business methods that execute the tasksassociated with the enterprise bean. For more information, see Implementing the business methods.
- It must define and implement an ejbCreate method for each way in which youwant it to be able to instantiate the enterprise bean class. For moreinformation, see Implementing the ejbCreate methods.
- It must be public, it must *not* be abstract, and it mustimplement the javax.ejb.SessionBean interface. For moreinformation, see Implementing the SessionBean interface.

**Note:**

Version 1.0 of the EJB specification allowed the methods in the sessionbean class to throw the java.rmi.RemoteException exception toindicate a non-application exception. This practice is deprecated inversion 1.1 of the specification. A session bean compliant withversion 1.1 of the specification should throw thejavax.ejb.EJBException exception (a subclass of thejava.lang.RuntimeException class) or another RuntimeExceptionexception instead. Because the javax.ejb.EJBExceptionclass is a subclass of the java.lang.RuntimeException,EJBException exceptions do not need to be explicitly listed in the throwsclause of methods.

A session bean can be either stateful or stateless. In a statelesssession bean, none of the methods depend on the values of variables set by anyother method, except for the ejbCreate method, which sets the initial(identical) state of each bean instance. In a stateful enterprise bean,one or more methods depend on the values of variables set by some othermethod. As in entity beans, static variables are not supported insession beans unless they are also final.Stateful session beans possibly need to synchronize their conversational statewith the transactional context in which they operate. For example, astateful session bean possibly needs to reset the value of some of itsvariables if a transaction is rolled back or it possibly needs to change thesevariables if a transaction successfully completes.

If a bean needs to synchronize its conversational state with thetransactional context, the bean class must implement thejavax.ejb.SessionSynchronization interface. Thisinterface contains methods to notify the session bean when a transactionbegins, when it is about to complete, and when it has completed. Theenterprise bean developer can use these methods to synchronize the state ofthe session enterprise bean instance with ongoing transactions.

**Note:**

The SessionSynchronization interface is *not* supported in the EJBserver (CB) environment.

The enterprise bean class can implement the enterprise bean's remoteinterface, but this is not recommended. If the enterprise bean classimplements the remote interface, it is possible to inadvertently pass the*this* variable as a method argument.

Figure 27 shows the main parts of the enterprise bean class for theexample Transfer bean. The sections that follow discuss these parts ingreater detail.

The Transfer bean is stateless. If the Transfer bean'stransferFunds method were dependent on the value of the *balance*variable returned by the getBalance method, the TransferBean would bestateful.

**Figure 27. Code example: The TransferBean class**

```
...import java.rmi.RemoteException;import java.util.Properties;import java.util.ResurceBundle;import
java.util.ListResourceBundle;import javax.ejb.*;import java.lang.*;import javax.naming.*;import
com.ibm.ejs.doc.account.*;...public class TransferBean implements SessionBean {     ...     private
SessionContext mySessionCtx = null;     private InitialContext initialContext = null;     private
AccountHome accountHome = null;     private Account fromAccount = null;     private Account
toAccount = null;     ...     public void ejbActivate() throws EJBException { }     ...     public
void ejbCreate() throws EJBException {          ...     }     ...     public void ejbPassivate()
throws EJBException { }     ...     public void ejbRemove() throws EJBException { }     ...
public float getBalance(long acctId) throws FinderException,          EJBException {          ...
}     ...     public void setSessionContext(javax.ejb.SessionContext ctx)          throws
EJBException {          ...     }     ...     public void transferFunds(long fromAcctId, long
toAcctId, float amount)          throws EJBException {          ...     }}
```

The business methods of a session bean class define the ways in which an EJBclient can manipulate the enterprise bean. The business methodsimplemented in the enterprise bean class cannot be directly invoked by an EJBclient. Instead, the EJB client invokes the corresponding methodsdefined in the enterprise bean's remote interface, by using an EJB objectassociated with an instance of the enterprise bean, and the container invokesthe corresponding methods in the enterprise bean instance.

Therefore, for every business method defined in the enterprise bean'sremote interface, a corresponding method must be implemented in the enterprisebean class. The enterprise bean's remote interface is implementedby the container in the EJBObject class when the enterprise bean isdeployed.

Figure 28 shows the business methods for the TransferBeanclass. The getBalance method is used to get the balance for anaccount. It first locates the appropriate Account EJB object and thencalls that object's getBalance method.

The transferFunds method is used to transfer a specified amount between twoaccounts (encapsulated in two Account entity EJB objects). Afterlocating the appropriate Account EJB objects by using the findByPrimaryKeymethod, the transferFunds method calls the add method on one account and thesubtract method on the other.Like all finder methods, findByPrimaryKey can throw both the FinderExceptionand RemoteException exceptions. The try/catch blocks are set up aroundinvocations of the findByPrimaryKey method to handle the entry of invalidaccount IDs by users. If the session bean user enters an invalidaccount ID, the findByPrimaryKey method cannot locate an EJB object, and thefinder method throws the FinderException exception. This exception iscaught and converted into a new FinderException exception containinginformation on the invalid account ID.

To call the findByPrimaryKey method, both business methods need to be ableto access the EJB home object that implements the AccountHome interfacediscussed in Writing the home interface (entity with CMP). Obtaining the EJB home object is discussed in Implementing the ejbCreate methods.

**Figure 28. Code example: The business methods of the TransferBean class**

```
public class TransferBean implements SessionBean {     ...   private Account fromAccount = null;
private Account toAccount = null;   ...   public float getBalance(long acctId) throws
FinderException, EJBException {        AccountKey key = new AccountKey(acctId);        try {
fromAccount = accountHome.findByPrimaryKey(key);        } catch(FinderException ex) {
throw new FinderException("Account " + acctId                          + " does not exist.");
} catch(RemoteException ex) {             throw new FinderException("Account " + acctId
+ " could not be found.");        }        return fromAccount.getBalance();   }   ...   public void
transferFunds(long fromAcctId, long toAcctId, float amount)         throws EJBException,
InsufficientFundsException, FinderException {        AccountKey fromKey = new
AccountKey(fromAcctId);        AccountKey toKey = new AccountKey(toAcctId);        try {
fromAccount = accountHome.findByPrimaryKey(fromKey);        } catch(FinderException ex) {
throw new FinderException("Account " + fromAcctId                          + " does not exist.");
} catch(RemoteException ex) {             throw new FinderException("Account " + acctId
+ " could not be found.");        }        try {             toAccount =
accountHome.findByPrimaryKey(toKey);        } catch(FinderException ex) {             throw new
FinderException("Account " + toAcctId                          + " does not exist.");        }
catch(RemoteException ex) {             throw new FinderException("Account " + acctId
+ " could not be found.");        }        try {             toAccount.add(amount);
fromAccount.subtract(amount);        } catch(InsufficientFundsException ex) {
mySessionCtx.setRollbackOnly();             throw new InsufficientFundsException("Insufficient
funds in  "                        + fromAcctId);        }   }}
```

#### Implementing the ejbCreate methods

You must define and implement an ejbCreate method for each way in which youwant an enterprise bean to be instantiated.

Each ejbCreate method must correspond to a create method in the enterprisebean's home interface. (Note that there is no ejbPostCreate methodin a session bean as there is in an entity bean.) Unlike the businessmethods of the enterprise bean class, the ejbCreate methods cannot be invokeddirectly by the client. Instead, the client invokes the create methodin the bean instance's home interface, and the container invokes theejbCreate method. If an ejbCreate method is executed successfully, anEJB object is created.

An ejbCreate method for a session bean must meet the followingrequirements:

- The method must be declared as public and cannot be declared as final orstatic.
- It must return void.
- A stateless session bean must have only one ejbCreate method, which mustreturn void and contain no arguments. A stateful session bean can havemultiple ejbCreate methods.

The throws clause can define arbitrary application exceptions. Thejavax.ejb.EJBException or another runtime exception can be usedto indicate non-application exceptions.

An ejbCreate method for an entity bean must meet the followingrequirements:

- The method must be declared as public and cannot be declared as final orstatic.
- It must return the entity bean's primary key type.
- It must contain code to set the values of any variables needed by the EJBobject.

The throws clause can define arbitrary application exceptions. Thejavax.ejb.EJBException or another runtime exception can be usedto indicate

non-application exceptions.Figure 29 shows the ejbCreate method required by the exampleTransferBean class. The Transfer bean's ejbCreate method obtains areference to the Account bean's home object. This reference isrequired by the Transfer bean's business methods. Getting areference to an enterprise bean's home interface is a two-stepprocess:

1. Construct an InitialContext object by setting the required propertyvalues. For the example Transfer bean, these property values aredefined in the environment variables of the Transfer bean's deploymentdescriptor.
2. Use the InitialContext object to create and get a reference to the homeobject. For the example Transfer bean, the JNDI name of the Accountbean is stored in an environment variable in the Transfer bean'sdeployment descriptor.

Creating the InitialContext object

When a container invokes the Transfer bean's ejbCreate method, theenterprise bean's *initialContext* object is constructed bycreating a Properties variable (*env*) that requires the followingvalues:

- The location of the name service(javax.naming.Context.PROVIDER_URL).
- The name of the initial context factory(javax.naming.Context.INITIAL_CONTEXT_FACTORY).

The values of these properties are discussed in more detail in Creating and getting a reference to a bean's EJB object.

**Figure 29. Code example: Creating the InitialContext object in the ejbCreate method of the TransferBean class**

```
...public class TransferBean implements SessionBean {     private static final String
INITIAL_NAMING_FACTORY_SYSPROP =                javax.naming.Context.INITIAL_CONTEXT_FACTORY;
private static final String PROVIDER_URL_SYSPROP =               javax.naming.Context.PROVIDER_URL;
...     private String nameService = null;     ...     private String providerURL = null;     ...
private InitialContext initialContext = null;     ...     public void ejbCreate() throws
EJBException {          // Get the initial context          try {              Properties env  =
System.getProperties();              ...              env.put( PROVIDER_URL_SYSPROP,
getProviderUrl() );              env.put( INITIAL_CONTEXT_FACTORY_SYSPROP, getNamingFactory() );
initialContext = new InitialContext( env );          } catch(Exception ex) {              ...
}          ...          // Look up the home interface using the JNDI name          ...}
```

Although the example Transfer bean stores some locale specific variables in aresource bundle class, like the example Account bean, it also relies on thevalues of environment variables stored in its deployment descriptor.Each of these InitialContext Properties values is obtained from an environmentvariable contained in the Transfer bean's deployment descriptor. Aprivate get method that corresponds to the property variable is used to geteach of the values (getNamingFactory and getProviderURL); these methodsmust be written by the enterprise bean developer. The followingenvironment variables must be set to the appropriate values in the deploymentdescriptor of the Transfer bean.

- javax.naming.Context.INITIAL_CONTEXT_FACTORY
- javax.naming.Context.PROVIDER_URL

(Setting environment variables for an enterprise bean shows an example of the **jetace** page required toset these variables.)

Figure 30 illustrates the relevant parts of the getProviderURL methodthat is used to get the PROVIDER_URL property value. Thejavax.ejb.SessionContext variable (*mySessionCtx*) isused to get the Transfer bean's environment in the deployment descriptorby invoking the getEnvironment method. The object returned by thegetEnvironment method can then be used to get the value of a specificenvironment variable by invoking the getProperty method.

**Figure 30. Code example: The getProviderURL method**

```
...public class TransferBean implements SessionBean {      private SessionContext mySessionCtx =
null;     ...     private String getProviderURL() throws RemoteException {          //get the
provider URL property either from          //the EJB properties or, if it isn't there          //use
"iiop:///", which causes a default to the local host          ...          String pr =
mySessionCtx.getEnvironment().getProperty(                    PROVIDER_URL_SYSPROP);          if (pr
== null)              pr = "iiop:///";          return pr;     }     ...}
```

Getting the reference to the home object

An enterprise bean is accessed by looking up the class implementing itshome interface by name through JNDI. Methods on the home interfaceprovide access to an instance of the class implementing the remoteinterface.

After constructing the InitialContext object, the ejbCreate method performsa JNDI lookup using the JNDI name of the Account enterprise bean. Likethe PROVIDER_URL and INITIAL_CONTEXT_FACTORY properties, this name is alsoretrieved from an environment variable contained in the Transfer bean'sdeployment descriptor (by invoking a private method named getHomeName).The lookup method returns an object of typejava.lang.Object.

The returned object is narrowed by using the static methodjavax.rmi.PortableRemoteObject.narrow to obtain areference to the EJB home object for the specified enterprise bean. Theparameters of the narrow method are the object to be narrowed and the class ofthe object to be created as a result of the narrowing. For a morethorough discussion of the code required to locate an enterprise bean in JNDIand then narrow it to get an EJB home object, see Creating and getting a reference to a bean's EJB object.

**Figure 31. Code example: Creating the AccountHome object in the ejbCreate method of the TransferBean class**

```
...public class TransferBean implements SessionBean {     ...     private String accountName = null;
...     private InitialContext initialContext = null;     ...     public void ejbCreate() throws
EJBException {          // Get the initial context          ...          // Look up the home
interface using the JNDI name          try {               java.lang.Object ejbHome =
initialContext.lookup(accountName);                              accountHome =
(AccountHome)javax.rmi.PortableRemoteObject.narrow(                         (org.omg.CORBA.Object)
ejbHome,  AccountHome.class);          } catch (NamingException e) { // Error getting the home
interface               ...          }          ...     }     ...}
```

Looking up an enterprise bean's environment naming context

The enterprise bean's environment is implemented by thecontainer. It enables the bean's business logic to be customizedwithout the need to access or change the bean's source code. Thecontainer provides an implementation of the JNDI naming context that storesthe enterprise bean environment. Business methods access theenvironment by using the JNDI interfaces. The deployment descriptorprovides the environment entries that the enterprise bean expects atruntime.

Each enterprise bean defines its own environment entries, which are sharedbetween all of its instances (that is, all instances with the samehome). Environment entries are not shared between enterprisebeans.

An enterprise bean's environment entries are stored directly in theenvironment naming context (or one of its subcontexts). To retrieve itsenvironment naming context, an enterprise bean instance creates anInitialContext object by using the constructor with no arguments. Itthen looks up the environment naming via the InitialContext object under thename java:comp/env.

The enterprise bean in Figure 32 changes an account number by looking up an environment entryto find the new account number.

**Figure 32. Code example: Looking up an enterprise bean's environment naming context**

```
public class AccountService implements SessionBean {...     public void changeAccountNumber(int
accountNumber, ... )          throws InvalidAccountNumberException{               ....
// Obtain the bean's environment naming context               Context initialContext = new
InitialContext();               Context myEnvironment =
(Context)initialContext.lookup("java:comp/env);            ...            // Obtain new account
number from environment            Integer newNumber =
(Integer)myEnvironment.lookup("newAccountNumber");          ... }}
```

#### Implementing the SessionBean interface

Every session bean class must implement the methods inherited from thejavax.ejb.SessionBean interface. The container invokesthese methods to inform the enterprise bean instance of significant events inthe instance's life cycle. All of these methods must be public,must return void, and can throw thejavax.ejb.EJBException. (Throwing thejava.rmi.RemoteException exception is deprecated; see *** for more information.)

- ejbActivate--This method is invoked by the container when thecontainer selects an enterprise bean instance from the instance pool andassigns it a specific existing EJB object. This method must contain anycode that you want to execute when the enterprise bean instance isactivated.
- ejbPassivate--This method is invoked by the container when thecontainer disassociates an enterprise bean instance from its EJB object andplaces the enterprise bean instance in the instance pool. This methodmust contain any code that you want to execute when the enterprise beaninstance is passivated (deactivated).
- ejbRemove--This method is invoked by the container when a clientinvokes the remove method inherited by the enterprise bean's homeinterface (from the javax.ejb.EJBHome interface). Thismethod must contain any code that you want to execute when an enterprise beaninstance is removed from the container.
- setSessionContext--This method is invoked by the container to pass areference to the javax.ejb.SessionContext interface to a sessionbean instance. If an enterprise bean instance needs to use this contextat any time during its life cycle, the enterprise bean class must contain aninstance variable to store this value. This method must contain anycode required to store a reference to the context.

  A session context can be used to get a handle to a particular instance of astateful session bean. It can also be used to get a reference to atransaction context object, as described in Using bean-managed transactions.

  **Note:**

  In the EJB server (CB) environment, the isCallerInRole and getCallerIdentitymethods inherited from the javax.ejb.EJBContext interface arenot supported.

As shown in Figure 33, except for the setSessionContext method, all of thesemethods in the TransferBean class are empty because no additional action isrequired by the bean for the particular life cycle states associated with thethese methods. The setSessionContext method is used in a conventionalway to set the value of the *mySessionCtx* variable.

**Figure 33. Code example: Implementing the SessionBean interface in the TransferBean class**

```
...public class TransferBean implements SessionBean {      private SessionContext mySessionCtx =
null;     ...     public void ejbActivate() throws EJBException { }     ...     public void
ejbPassivate() throws EJBException { }     ...     public void ejbRemove() throws EJBException { }
...     public void setSessionContext(SessionContext ctx) throwEJBException {          mySessionCtx
= ctx;      }     ...}
```

## Writing the home interface (session)

A session bean's home interface defines the methods used by clients tocreate and remove instances of the enterprise bean and obtain metadata aboutan instance. The home interface is defined by the enterprise beandeveloper and implemented in the EJB home class created by the containerduring enterprise bean deployment. The container makes the homeinterface accessible to clients through JNDI.

By convention, the home interface is named *Name*Home, where*Name* is the name you assign to the enterprise bean. Forexample, the Transfer enterprise bean's home interface is namedTransferHome.Every session bean's home interface must meet the followingrequirements:

- It must extend the javax.ejb.EJBHome interface. Thehome interface inherits several methods from thejavax.ejb.EJBHome interface. See The javax.ejb.EJBHome interface for information on thesemethods.
- Each method in the interface must be a create method that corresponds to aejbCreate method in the enterprise bean class. For more information,see Implementing the ejbCreate methods. Unlike entity beans, the home interface of a sessionbean contains no finder methods.
- The parameters and return value of each method defined in the interfacemust be valid for Java RMI. For more information, see The java.io.Serializable and java.rmi.Remote interfaces. In addition, each method's throws clause mustinclude the java.rmi.RemoteException exception class.

Figure 34 shows the relevant parts of the definition of the homeinterface (TransferHome) for the example Transfer bean.

**Figure 34. Code example: The TransferHome home interface**

```
...import javax.ejb.*;import java.rmi.*;public interface TransferHome extends EJBHome {     Transfer
create() throws CreateException, RemoteException;  }
```

A create method is used by a client to create an enterprise beaninstance. A stateful session bean can contain multiple createmethods; however, a stateless session bean can contain only one createmethod with no arguments. This restriction on stateless session beansensures that every instance of a stateless session bean is the same as everyother instance of the same type. (For example, every Transfer beaninstance is the same as every other Transfer bean instance.)

Each create method must be named create and have the same number and typesof arguments as a corresponding ejbCreate method in the EJB objectclass. The return types of the create method and its correspondingejbCreate method are always different.Each create method must meet the following requirements:

- It must return the type of the enterprise bean's remoteinterface. For example, the return type for the create method in theTransferHome interface is Transfer.
- It must have a throws clause that includes thejava.rmi.RemoteException exception, thejavax.ejb.CreateException exception class, and all of theexceptions defined in the throws clause of the corresponding ejbCreatemethod.

## Writing the remote interface (session)

A session bean's remote interface provides access to the business methodsavailable in the enterprise bean class. It also provides methods toremove an enterprise bean instance and to obtain the enterprise bean'shome interface and handle. The remote interface is defined by theenterprise bean developer and implemented in the EJB object class created bythe container during enterprise bean deployment.

By convention, the remote interface is named *Name*, where*Name* is the name you assign to the enterprise bean. Forexample, the Transfer enterprise bean's remote interface is namedTransfer.Every remote interface must meet the following requirements:

- It must extend the javax.ejb.EJBObject interface. Theremote interface inherits several methods from the EJBObject interface.See Methods inherited from javax.ejb.EJBObject for information on these methods.
- You must define a corresponding business method for every business methodimplemented in the enterprise bean class.
- The parameters and return value of each method defined in the interfacemust be valid for Java RMI. For more information, see The java.io.Serializable and java.rmi.Remote interfaces.
- Each method's throws clause must include thejava.rmi.RemoteException exception class.

Figure 35 shows the relevant parts of the definition of the remoteinterface (Transfer) for the example Transfer bean. This interfacedefines the methods for transferring funds between two Account bean instancesand for getting the balance of an Account bean instance.

**Figure 35. Code example: The Transfer remote interface**

```
...import javax.ejb.*;import java.rmi.*;import com.ibm.ejs.doc.account.*;public interface Transfer
extends EJBObject {     ...     float getBalance(long acctId) throws FinderException,
RemoteException;     ...     void transferFunds(long fromAcctId, long toAcctId, float amount)
throws InsufficientFundsException, RemoteException;}
```

---

## Implementing interfaces common to multiple types of enterprise beans

Enterprise beans must implement the interfaces described here in theappropriate enterprise bean component.

### Methods inherited from javax.ejb.EJBObject

The remote interface inherits the following methods from thejavax.ejb.EJBObject interface, which are implemented by thecontainer during deployment:

- getEJBHome--Returns the enterprise bean's home interface.
- getHandle--Returns the handle for the EJB object.
- getPrimaryKey--Returns the EJB object's primary key. (Forsession beans, this cannot be used because session beans do not have a primarykey.)
- isIdentical--Compares this EJB object with the EJB object argument todetermine if they are the same.
- remove--Removes this EJB object.

These methods have the following syntax:

```
public abstract EJBHome getEJBHome();public abstract Handle getHandle();public abstract Object
getPrimaryKey();public abstract boolean isIdentical(EJBObject obj);public abstract void remove();
```

These methods are implemented by the container in the EJB objectclass.

### The javax.ejb.EJBHome interface

The home interface inherits two remove methods and the getEJBMetaData methodfrom the javax.ejb.EJBHome interface. Just like themethods defined directly in the home interface, these inherited methods arealso implemented in the EJB home class created by the container duringdeployment.

The remove methods are used to remove an existing EJB object (and itsassociated data in the database) either by specifying the EJB object'shandle or its primary key. (The remove method that takes a*primaryKey* variable can be used only in entity beans.) ThegetEJBMetaData method is used to obtain metadata about the enterprise bean andis mainly intended for use by development tools.

These methods have the following syntax:

```
public abstract EJBMetaData getEJBMetaData();public abstract void remove(Handle handle);public
abstract void remove(Object primaryKey);
```

The javax.ejb.EJBHome interface also contains a method to geta handle to the home interface. It has the following syntax:

```
public abstract HomeHandle getHomeHandle();
```

### The java.io.Serializable and java.rmi.Remote interfaces

To be valid for use in a remote method invocation (RMI), a method'sarguments and return value must be one of the following types:

- A primitive type; for example, an int or a long.
- An object of a class that directly or indirectly implementsjava.io.Serializable; for example,java.lang.Long.
- An object of a class that directly or indirectly implementsjava.rmi.Remote.
- An array of valid types or objects.

If you attempt to use a parameter that is not valid, thejava.rmi.RemoteException exception is thrown. Note thatthe following atypical types are *not* valid:

- An object of a class that directly or indirectly implements bothSerializable and Remote.
- An object of a class that directly or indirectly implements Remote, butcontains a method that does not throw the RemoteException or an exception thatinherits from RemoteException.

---

## Using threads and reentrancy in enterprise beans

An enterprise bean must not contain code to start new threads (nor can methodsbe defined with the keyword synchronized). Session beans can*never* be reentrant; that is, they cannot call another beanthat invokes a method on the calling bean. Entity beans can bereentrant, but building reentrant entity beans is not recommended and is notdocumented here.

The EJB server (AE) enforces single-threaded access to all enterprisebeans. Illegal callbacks result in ajava.rmi.RemoteException exception being thrown to the EJBclient.

The EJB server (CB) enforces single-threaded access to enterprise beansonly if their transaction attribute is set to either TX_NOT_SUPPORTED orTX_BEAN_MANAGED. For other enterprise beans, access from differenttransactions is serialized, but serialized access from different threadsrunning under the same transaction is not enforced. For enterprisebeans deployed with the transaction attribute value of TX_NOT_SUPPORTED orTX_BEAN_MANAGED, illegal callbacks result in a RemoteException exception beingthrown to the EJB client.

---

## Creating an EJB module for enterprise beans

There are two tasks involved in preparing an enterprise bean fordeployment:

- Making the components of the bean part of the same Java package.For more information, see Making bean components part of a Java package.
- Creating an EJB module and associated deployment descriptor (AEonly). For more information, see Creating an EJB module and deployment descriptor.

If you develop enterprise beans in an IDE, these tasks are handled fromwithin the tool that you use. If you do not develop enterprise beans inan IDE, you must handle each of these tasks by using tools contained in theJava Software Development Kit (SDK) and WebSphere Application Server.

- For more information on the tools used to create an EJB module in the EJBserver (AE) programming environment, see Tools for developing and deploying enterprise beans in the EJB server (AE) environment.
- For more information on the tools used to package beans in the EJB server(CB) programming environment, see Tools for developing and deploying enterprise beans in the EJB server (CB) environment.

### Making bean components part of a Java package

You determine the best way to allocate your enterprise beans to Javapackages. A Java package can contain one or more enterprisebeans. The example Account and Transfer beans are stored in separatepackages. All of the Java source files that make up the Account beancontain the following package statement:

```
package com.ibm.ejs.doc.account;
```

All of the Java source files that make up the Transfer bean contain thefollowing package statement:

```
package com.ibm.ejs.doc.transfer;
```

### Creating an EJB module and deployment descriptor

An EJB module contains one or more deployable enterprise beans. Italso contains a deployment descriptor that provides information about eachenterprise bean and instructions for the container on how to handle allenterprise beans in the module. The deployment descriptor is stored inan XML file.

During creation of the EJB module, you specify the files for eachenterprise bean to be included in the module. These filesinclude:

- The class files associated with each component of the enterprisebean.
- Any additional classes and files associated with the enterprise bean;for example: user-defined exception classes, properties files, andresource bundle classes.

You also specify other information about the bean, such as references toother enterprise beans, resource factories, and security roles. Afterdefining the enterprise beans to be included in the module, you specifyapplication assembly instructions that apply to the module as a whole.Both bean and module information are used to create a deploymentdescriptor. See The deployment descriptor for a list of deployment descriptor settings andattributes.

# Developing EJB clients

An enterprise bean can be accessed by all of the following types of EJBclients in both EJB server environments:

- Java servlets. For more information about writing Java servletsthat use enterprise beans, see Developing servlets that use enterprise beans.
- Java Server Pages (JSP). For more information about writing JSP,consult a commercially available book.
- Java applications that use remote method invocation (RMI). For moreinformation on writing Java applications, consult a commercially availablebook.
- Other enterprise beans. For example, the Transfer session bean actsas a client to the Account bean, as described in Developing enterprise beans.

It is recommended that you avoid accessing EJB entity beans from client orservlet code. Instead, wrap and access EJB entity beans from EJBsession beans. This improves performance in two ways:

- It reduces the number of remote method calls. When the clientapplication accesses the entity bean directly, each getter method is a remotecall. A wrapping session bean can access the entity bean locally, andcollect the data in a structure, which it returns by value.
- It provides an outer transaction context for the EJB entity bean.An entity bean synchronizes its state with its underlying data store at thecompletion of each transaction. When the client application accessesthe entity bean directly, each getter method becomes a completetransaction. A store and a load action follow each method. Whenthe session bean wraps the entity bean to provide an outer transactioncontext, the entity bean synchronizes its state when the outer session beanreaches a transaction boundary.

Except for the basic programming tasks described in this chapter, creatinga Java servlet, JSP, or Java application that is a client to an enterprisebean is not very different from designing standard versions of these types ofJava programs. This chapter assumes that you understand the basics ofwriting a Java servlet, a Java application, or a JSP file.

Except where noted, all of the code described in this chapter is taken fromthe example Java application named TransferApplication. This Javaapplication and the other EJB clients available with the documentation examplecode are explained in Information about the examples described in the documentation.

To access and manipulate an enterprise bean in any of the Java-based EJBclient types listed previously, the EJB client must do the following:

- Import the Java packages required for naming, remote method invocation(RMI), and enterprise bean interaction.
- Get a reference to an instance of the bean's EJB object by using theJava Naming and Directory Interface (JNDI). For more information, see Creating and getting a reference to a bean's EJB object.
- Handle invalid EJB objects when using session beans. For moreinformation, see Handling an invalid EJB object for a session bean.
- Remove session EJB objects when they are no longer required or removeentity EJB objects when the associated data in the data source must beremoved. For more information, see Removing a bean's EJB object.

In addition, an EJB client can participate in the transactions associatedwith enterprise beans used by the client. For more information, see Managing transactions in an EJB client.

**Note:**

In the EJB server (CB) environment, an enterprise bean can also be accessedby a Java applet, an ActiveX client, a CORBA-based Java client, and to alimited degree, by a C++ CORBA client. The Travel example brieflydescribed in Information about the examples described in the documentation illustrates some of these types of clients. More information on EJB clients specific to the EJB server (CB) provides additional information about EJBclients that use ActiveX and CORBA-based Java and C++.

---

## Importing required Java packages

Although the Java packages required for any particular EJB client vary, thefollowing packages are required by all EJB clients:

- java.rmi -- This package contains most of the classes requiredfor remote method invocation (RMI).
- javax.rmi -- This package contains the PortableRemoteObjectclass required to get a reference to an EJB object.
- java.util -- This package contains various Java utilityclasses, such as Properties, Hashtable, and Enumeration used in a variety ofways throughout all enterprise beans and EJB clients.
- javax.ejb -- This package contains the classes and interfacesdefined in the EJB specification.
- javax.naming -- The package contains the classes andinterfaces defined in the Java Naming and Directory Interface (JNDI)specification and is used by clients to get references to EJB objects.
- The package or packages containing the enterprise beans with which theclient interacts.

The Java client object request broker (ORB), which is automaticallyinitialized in EJB clients, does not support dynamic download ofimplementation bytecode from the server to the client. As a result, allclasses required by the EJB client at runtime must be available from the filesand directories identified in the client's CLASSPATH environmentvariable. For information on the JAR files required by EJB clients, seeSetting the CLASSPATH environment variable in the EJB server (AE) environment or Setting the CLASSPATH environment variable in the EJB server (CB) environment. You can install needed files onyour client machine by doing a WebSphere Application Server installation onthe machine. If you are using the Advanced Application Server, selectthe **Developer's Client Files** option; if you are usingComponent Broker, select the **Java client** option. You alsoneed to make sure that the ioser and ioserx executable files are accessible onyour client machine; these files are normally part of the Javainstall. If you are using Windows NT, make sure that EJB clients canlocate the ioser.dll library file at run time.Figure 36 shows the import statements for the example Java applicationcom.ibm.ejs.doc.client.TransferApplication.In addition to the required Java packages mentioned previously, the exampleapplication imports the com.ibm.ejs.doc.transferpackage because the application communicates with a Transfer bean. Theexample application also imports the InsufficientFundsException classcontained in the same package as the Account bean.

**Figure 36. Code example: The import statements for the Java application TransferApplication**

```
...import java.awt.*;import java.awt.event.*;import java.util.*;import java.rmi.*...import
javax.naming.*;import javax.ejb.*;import javax.rmi.PortableRemoteObject;...import
com.ibm.ejs.doc.account.InsufficientFundsException;import com.ibm.ejs.doc.transfer.*;...public class
TransferApplication extends Frame implements      ActionListener, WindowListener {     ...}
```

## Creating and getting a reference to a bean's EJB object

To invoke a bean's business methods, a client must create or find an EJBobject for that bean. After the client has created or found thisobject, it can invoke methods on it in the standard way.

To create or find an instance of a bean's EJB object, the client mustdo the following:

1. Locate and create an EJB home object for that bean. For moreinformation, see Locating and creating an EJB home object.
2. Use the EJB home object to create or (for entity beans only) find aninstance of the bean's EJB object. For more information, see Creating an EJB object.

The TransferApplication client contains one reference to a Transfer EJBobject, which the application uses to invoke all of the methods on theTransfer bean. When using session beans in Java applications, it is agood idea to make the reference to the EJB object a class-level variablerather than a variable that is local to a method. This allows your EJBclient to repeatedly invoke methods on the same EJB object rather than havingto create a new object each time the client invokes a session beanmethod. As discussed in Threading issues, this approach is not recommended for servlets, whichmust be designed to handle multiple threads.

### Locating and creating an EJB home object

JNDI is used to find the name of an EJB home object. The propertiesthat an EJB client uses to initialize JNDI and find an EJB home object varyacross EJB server implementations. To make an enterprise bean moreportable between EJB server implementations, it is recommended that youexternalize these properties in environment variables, properties files, orresource bundles rather than hard code them into your enterprise bean or EJBclient code.

The example Transfer bean uses environment variables as discussed in Implementing the ejbCreate methods. The TransferApplication uses a resource bundlecontained in thecom.ibm.ejs.doc.client.ClientResourceBundle.classfile.To initialize a JNDI name service, an EJB client must set the appropriatevalues for the following JNDI properties:

**javax.naming.Context.PROVIDER_URL**

This property specifies the host name and port of the name server used bythe EJB client. The property value must have the followingformat: iiop://*hostname*:*port*, where*hostname* is the IP address or hostname of the machine on which thename server runs and *port* is the port number on which the nameserver listens.

For example, the property value`iiop://bankserver.mybank.com:9019` directsan EJB client to look for a name server on the host namedbankserver.mybank.com listening on port 9019. Theproperty value `iiop://bankserver.mybank.com`directs an EJB client to look for a name server on the host namedbankserver.mybank.com at port number 900. The propertyvalue `iiop:///` directs an EJB client to look for a nameserver on the local host listening on port 900. If not specified, thisproperty defaults to the local host and port number 900, which is the same asspecifying `iiop:///`. In the EJB server (AE), the portnumber used by the name service can be changed by using the administrativeinterface.

**javax.naming.Context.INITIAL_CONTEXT_FACTORY**

This property identifies the actual name service that the EJB client mustuse.

- In the EJB server (AE) environment, this property must be set to`com.ibm.ejs.ns.jndi.CNInitialContextFactory`.
- In the EJB server (CB) environment, this property must be set to`com.ibm.ejb.cb.runtime.CBCtxFactory`,to one of its subclasses (such as`com.ibm.ejb.cb.runtime.CBCtxFactoryHostDefault`,or to an initial context factory created by using the **appbind**tool. When using this context factory, thejavax.naming.Context.list andjavax.naming.Context.listBindings methods can return nomore than 1000 elements in the javax.naming.NamingEnumerationobject. For more information on using the **appbind** tool, see Application-specific contexts and the appbind tool.

Locating an EJB home object is a two-step process:

1. Create a javax.naming.InitialContext object. For moreinformation, see Creating an InitialContext object.
2. Use the InitialContext object to create the EJB home object. Formore information, see Creating EJB home object.

#### Creating an InitialContext object

Figure 37 shows the code required to create the InitialContextobject. To create this object, construct ajava.util.Properties object, add values to the Propertiesobject, and then pass the object as the argument to the InitialContextconstructor. In the TransferApplication, the value of each property isobtained from the resource bundle class namedcom.ibm.ejs.doc.client.ClientResourceBundle,which stores all of the locale-specific variables required by theTransferApplication. (This class also stores the variables used by theother EJB clients contained in the documentation example, described in Information about the examples described in the documentation).The resource bundle class is instantiated by calling theResourceBundle.getBundle method. The values of variables withinthe resource bundle class are extracted by calling the getString method on the*bundle* object.

The createTransfer method of the TransferApplication can be called multipletimes as explained in Handling an invalid EJB object for a session bean. However, after the InitialContext object is createdonce, it remains good for the life of the client session. Therefore,the code required to create the InitialContext object is placed within an ifstatement that determines if the reference to the InitialContext object isnull. If the reference is null, the InitialContext object iscreated; otherwise, the reference can be reused on subsequent creationsof the EJB object.

**Figure 37. Code example: Creating the InitialContext object**

```
...public class TransferApplication extends Frame implements ActionListener,      WindowListener {
...     private InitialContext ivjInitContext = null;     private Transfer ivjTransfer = null;
private ResourceBundle bundle = ResourceBundle.getBundle(
"com.ibm.ejs.doc.client.ClientResourceBundle");     ...     private String nameService = null;
private String accountName = null;     private String providerUrl = null;     ...     private
Transfer createTransfer() {          TransferHome transferHome = null;          Transfer transfer =
null;     // Get the initial context     if (ivjInitContext == null) {          try {
Properties properties = new Properties();               // Get location of name service
properties.put(javax.naming.Context.PROVIDER_URL,
bundle.getString("providerUrl"));               // Get name of initial context factory
properties.put(javax.naming.Context.INITIAL_CONTEXT_FACTORY,
bundle.getString("nameService"));               ...               ivjInitContext = new
InitialContext(properties);          } catch (Exception e) { // Error getting the initial context
...          }     }     ...     // Look up the home interface using the JNDI name     ...     //
Create a new Transfer object to return     ...     return transfer;}
```

#### Creating EJB home object

After the InitialContext object (*ivjInitContext)* is created, theapplication uses it to create the EJB home object, as shown in Figure 38. This creation is accomplished by invoking the lookupmethod, which takes the JNDI name of the enterprise bean in String form andreturns a java.lang.Object object:

- When performing a JNDI lookup on an enterprise bean deployed in an EJBserver (AE; CB on AIX, Windows NT, or Solaris platforms), only the JNDIname specified in the deployment descriptor is used.
- When performing a JNDI lookup on an enterprise bean deployed in an EJBserver (CB on platforms other than AIX, Windows NT, and Solaris), the JNDIhome name passed to the lookup method is the JNDI name specified in theenterprise bean's deployment descriptor with a CB-specific prefixattached. The content of this prefix depends on where in the ComponentBroker namespace the system administrator bound the EJB home (by using the**ejbbind** tool).

  If the system administrator binds the EJB home in the host name tree of aspecific bootstrap host, then the JNDI name prefix will be`host/resources/factories/EJBHomes`. If the systemadministrator binds the EJB home in a workgroup name tree, then the JNDI nameprefix will be `workgroup/resources/factories/EJBHomes`, and the EJBclient must belong to the same preferred workgroup. If the systemadministrator binds the EJB home in the cell name tree, then the JNDI nameprefix is `cell/resources/factories/EJBHomes`.

The example TransferApplication gets the JNDI name of the Transfer beanfrom the ClientResourceBundle class.After an object is returned by the lookup method, the static methodjavax.rmi.PortableRemoteObject.narrow is used to obtainan EJB home object for the specified enterprise bean. The narrow methodtakes two parameters: the object to be narrowed and the class of the EJBhome object to be returned by the narrow method. The object returned bythe javax.rmi.PortableRemoteObject.narrow method is castto the class associated with the home interface.

**Figure 38. Code example: Creating the EJBHome object**

```
private Transfer createTransfer() {     TransferHome transferHome = null;     Transfer transfer =
null;     // Get the initial context     ...     // Look up the home interface using the JNDI name
try {          java.lang.Object homeObject = ivjInitContext.lookup(
bundle.getString("transferName"));          transferHome =
(TransferHome)javax.rmi.PortableRemoteObject.narrow(                    (org.omg.CORBA.Object)
homeObject, TransferHome.class);     } catch (Exception e)  { // Error getting the home interface
...     }      ...     // Create a new Transfer object to return     ...     return transfer;}
```

## Creating an EJB object

After the EJB home object is created, it is used to create the EJBobject. Figure 39 shows the code required to create the EJB object byusing the EJB home object. A create method is invoked to create an EJBobject or (for entity beans only) a finder method is invoked to find anexisting EJB object. Because the Transfer bean is a stateless sessionbean, the only choice is the default create method.

**Figure 39. Code example: Creating the EJB object**

```
private Transfer createTransfer() {     TransferHome transferHome = null;     Transfer transfer =
null;     // Get the initial context     ...     // Look up the home interface using the JNDI name
...     // Create a new Transfer object to return     try {          transfer =
transferHome.create();     } catch (Exception e) { // Error creating Transfer object          ...
}     ...     return transfer;}
```

---

## Handling an invalid EJB object for a session bean

Because session beans are ephemeral, the client cannot depend on a sessionbean's EJB object to remain valid. A reference to an EJB objectfor a session bean can become invalid if the EJB server fails or is restartedor if the session bean times out due to inactivity. (The reference toan entity bean's EJB object is always valid until that object isremoved.) Therefore, the client of a session bean must contain code tohandle a situation in which the EJB object becomes

invalid.

An EJB client can determine if an EJB object is valid by placing all methodinvocations that use the reference inside of a try/catch block thatspecifically catches the java.rmi.NoSuchObjectException, inaddition to any other exceptions that the method needs to handle. TheEJB client can then invoke the code to handle this exception.

You determine how to handle an invalid EJB object. The exampleTransferApplication creates a new Transfer EJB object if the one it iscurrently using becomes invalid.The code to create a new EJB object when the old one becomes invalid is thesame code used to create the original EJB object and is described in Creating and getting a reference to a bean's EJB object. For the example TransferApplication client, thiscode is contained in the createTransfer method.

Figure 40 shows the code used to create the new EJB object in thegetBalance method of the example TransferApplication. The getBalancemethod contains the local boolean variable *sessionGood*, which isused to specify the validity of the EJB object referenced by the variable*ivjTransfer*. The *sessionGood* variable is alsoused to determine when to break out of the do-while loop.The *sessionGood* variable is initialized to false because the*ivjTransfer* can reference an invalid EJB object when the getBalancemethod is called. If the *ivjTransfer* reference is valid, theTransferApplication invokes the Transfer bean's getBalance method andreturns the balance. If the *ivjTransfer* reference isinvalid, the NoSuchObjectException is caught, the TransferApplication'screateTransfer method is called to create a new Transfer EJB object reference,and the *sessionGood* variable is set to false so that the do-whileloop is repeated with the new valid EJB object. To prevent an infiniteloop, the *sessionGood* variable is set to true when any otherexception is thrown.

**Figure 40. Code example: Refreshing the EJB object reference for a session bean**

```
private float getBalance(long acctId) throws NumberFormatException, RemoteException,
FinderException {     // Assume that the reference to the Transfer session bean is no good     ...
boolean sessionGood = false;     float balance = 0.0f;     do {          try {                //
Attempt to get a balance for the specified account              balance =
ivjTransfer.getBalance(acctId);               sessionGood = true;               ...          }
catch(NoSuchObjectException ex) {               createTransfer();               sessionGood = false;
} catch(RemoteException ex) {              // Server or connection problem               ...
} catch(NumberFormatException ex) {               // Invalid account number               ...
} catch(FinderException ex) {               // Invalid account number              ...          }
} while(!sessionGood);     return balance;}
```

## Removing a bean's EJB object

When an EJB client no longer needs a stateful session EJB object, the EJBclient should remove that object. Instances of stateful session beanshave affinity to specific clients. They will remain in the containeruntil they are explicitly removed by the client, or removed by the containerwhen they time out. Meanwhile, the container might need to passivateinactive stateful session beans to disk. This requires overhead for thecontainer and impacts performance of the application. If the passivatedsession bean is subsequently required by the application, the containeractivates it by restoring it from disk. By explicitly removing statefulsession beans when finished with them, applications can decrease the need forpassivation and minimize container overhead.

You remove entity EJB objects *only* when you want to remove theinformation in the data source with which the entity EJB object isassociated.

To remove an EJB object, invoke the remove method on the object. Asdiscussed in Creating and getting a reference to a bean's EJB object, the TransferApplication contains only one reference to aTransfer EJB object that is created when the application isinitialized.

Figure 41 shows how the example Transfer EJB object is removed in theTransferApplication in the killApp method. To parallel the creation ofthe Transfer EJB object when the TransferApplication is initialized, theapplication removes the final EJB object associated with*ivjTransfer* reference right before closing the application'sGUI window. The killApp method closes the window by invoking thedispose method on itself.

**Figure 41. Code example: Removing a session EJB object**

```
...private void killApp() {     try {          ivjTransfer.remove();          this.dispose();
System.exit(0);     } catch (Throwable ivjExc) {          ...     }}
```

## Managing transactions in an EJB client

In general, it is practical to design your enterprise beans so that alltransaction management is handled at the enterprise bean level. In astrict three-tier, distributed application, this is not always possible oreven desirable. However, because the middle tier of an EJB applicationcan include two subcomponents--session beans and entity beans--it ismuch easier to design the transactional management completely within theapplication server tier. Of course, the resource manager tier must alsobe designed to support transactions.

**Note:**

> EJB clients that access entity beans with CMP that use Host On-Demand (HOD)or the External Call Interface (ECI) for CICS or IMS applications must begin atransaction before invoking a method on these entity beans. Thisrestriction is required because these types of entity beans must use theTX_MANDATORY transaction attribute.

Nevertheless, it is still possible to program an EJB client (that is not anenterprise bean) to participate in transactions for those specializedsituations that require it. To participate in a transaction, the EJBclient must do the following:

1. Obtain a reference to the javax.transaction.UserTransactioninterface by using JNDI as defined in the Java Transaction ApplicationProgramming

Interface (JTA).

2. Use the object reference to invoke any of the following methods:
    - begin--Begins a transaction. This method takes no argumentsand returns void.
    - commit--Attempts to commit a transaction; assuming that nothingcauses the transaction to be rolled back, successful completion of this methodcommits the transaction. This method takes no arguments and returnsvoid.
    - getStatus--Returns the status of the referenced transaction.This method takes no arguments and returns int; if no transaction isassociated with the reference, STATUS_NO_TRANSACTION is returned. Thefollowing are the valid return values for this method:
        - STATUS_ACTIVE--Indicates that transaction processing is still inprogress.
        - STATUS_COMMITTED--Indicates that a transaction has been committed andthe effects of the transaction have been made permanent.
        - STATUS_COMMITTING--Indicates that a transaction is in the process ofcommitting (that is, the transaction has started committing but has notcompleted the process).
        - STATUS_MARKED_ROLLBACK--Indicates that a transaction is marked to berolled back.
        - STATUS_NO_TRANSACTION--Indicates that a transaction does not exist inthe current transaction context.
        - STATUS_PREPARED--Indicates that a transaction has been prepared butnot completed.
        - STATUS_PREPARING--Indicates that a transaction is in the process ofpreparing (that is, the transaction has started preparing but has notcompleted the process).
        - STATUS_ROLLEDBACK--Indicates that a transaction has been rolledback.
        - STATUS_ROLLING_BACK--Indicates that a transaction is in the processof rolling back (that is, the transaction has started rolling back but has notcompleted the process).
        - STATUS_UNKNOWN--Indicates that the status of a transaction isunknown.
    - rollback--Rolls back the referenced transaction. This methodtakes no arguments and returns void.
    - setRollbackOnly--Specifies that the only possible outcome of thetransaction is for it to be rolled back. This method takes no argumentsand returns void.
    - setTransactionTimeout--Sets the timeout (in seconds) associated withthe transaction. If some transaction participant has not specificallyset this value, a default timeout is used. This method takes a numberof seconds (as type int) and returns void.

Figure 42 provides an example of an EJB client creating a reference toa UserTransaction object and then using that object to set the transactiontimeout, begin a transaction, and attempt to commit the transaction.(The source code for this example is *not* available with the examplecode provided with this document.) Notice that the client does a simpletype cast of the lookup result, rather than invoking a narrow method asrequired with other JNDI lookups. In both EJB server environments, theJNDI name of the UserTransaction interface is`java:comp/UserTransaction`.

**Figure 42. Code example: Managing transactions in an EJB client**

```
...import javax.transaction.*;...// Use JNDI to locate the UserTransaction objectContext
initialContext = new InitialContext();UserTransaction tranContext = (
UserTransaction)initialContext.lookup("java:comp/UserTransaction");// Set the transaction timeout to
30 secondstranContext.setTransactionTimeout(30);...// Begin a transactiontranContext.begin();//
Perform transaction work invoking methods on enterprise bean references...// Call for the
transaction to committranContext.commit();
```

## More information on EJB clients specific to the EJB server (CB)

When developing EJB clients for the EJB server (CB) environment, you candevelop the following types of clients:

- Microsoft ActiveX clients. For some general information, see EJB clients that use ActiveX.
- Clients using the Component Broker Session Service. For somegeneral information, see Clients using the Component Broker Session Service.

For more information on developing these types of clients, see the IBMRedbook entitled *IBM Component Broker Connector Overview*, formnumber SG24-2022-02.

### EJB clients that use ActiveX

If you write your EJB client as a component that adheres to the JavaBeans(TM)Specification, you can use the JavaBeans bridge to run the EJB client as anActiveX control. An EJB client of this type must provide a no-argumentconstructor, it must implement the java.io.Serializableinterface, and it must have a readObject and a writeObject method, ifapplicable.

If your EJB client is also an applet, you must not perform your JNDIinitialization as part of object construction. Rather, perform JNDIinitialization in the applet's start method. The JavaBeans bridgemust create an instance of your EJB client so that it can introspect it andmake the necessary stubs to create the ActiveX proxy for it. You mustdelay the JNDI connections until the user can specify the necessary propertiesby way of the ActiveX property sheet.

### Clients using the Component Broker Session Service

In addition to the Transaction Service, Component Broker also provides aSession Service for the Procedural Application Adaptor (PAA) that enables theuse of backend systems such as CICS and IMS. Since the JTA does nothave a Session Service, it is not possible to use JNDI to look up a handle tothe service in an EJB client. In this case, the EJB client must act asan ordinary CB Java client.

The normal lookup procedure for a CB Java client is to use the CORBAresolve_initial_references method. In this case, the CORBA object tolook up is named SessionCurrent.

Before you can call the resolve_initial_references method, the ORB needs tobe properly initialized for the CB runtime environment. Theinitialization method depends on whether or not you are using VisualAge forJava access beans in the CB environment. If you are using access beans,then the ORB must be manually initialized. ORB initialization in accessbeans is done in a "lazy" fashion. That is, initialization is not doneuntil the first remote method is invoked. However, because a sessionmust be started before that method is called, the ORB initialization must bedone manually. The example code in Figure 43 shows this initialization.

**Figure 43. Code example: Initializing the ORB (if using access beans)**

```
String[] CBargs = null;CBargs = new String[6];CBargs[0] = "-ORBBootstrapHost";  // substitute your
bootstrap host nameCBargs[1] = "cbs3.rchland.ibm.com"; CBargs[2] = "-ORBBootstrapPort";CBargs[3] =
"900";CBargs[4] = "-ORBClass";CBargs[5] =
"com.ibm.CORBA.iiop.ORB";com.ibm.CBCUtil.CBSeriesGlobal.Initialize(CBargs);
```

If you are not using access beans, initialization code is notnecessary. The ORB is properly initialized during the creation of theInitialContext object with the appropriate properties. For example,your client code should already contain lines similar to those in Figure 44. This code is used to find the service, look up thehome object, narrow the home object, and create the proxy object (tasksautomatically done if an access bean is being used).

**Figure 44. Code example: Creating the InitialContext object (if not using access beans)**

```
Properties properties = new Properties();properties.put(javax.naming.Context.PROVIDER_URL,
"iiop:///");// CB Factory
Nameproperties.put(javax.naming.Context.INITIAL_CONTEXT_FACTORY,"com.ibm.ejb.cb.runtime.CBCtxFactory");
Context ctx = new InitialContext(properties);
```

After the ORB is initialized (either automatically or manually), you mustuse CB-specific APIs for creating and using the sessionCurrent object.You must include code similar to the example code in Figure 45.

**Figure 45. Code example: Creating and using the sessionCurrent object**

```
org.omg.CORBA.Object orbCurrent = null;com.ibm.ISessions.Current sessionCurrent = null;...orbCurrent
= com.ibm.CBCUtil.CBSeriesGlobal.orb().resolve_initial_references(
"ISessions::Current");sessionCurrent =
com.ibm.ISessions.CurrentHelper.narrow(orbCurrent);sessionCurrent.beginSession("myApp");...//
commitsessionCurrent.endSession(com.ibm.ISessions.EndMode.EndModeCheckPoint, true);
```

For more information on using the resolve_initial_references method, seethe Component Broker Programming Guide.

# An architectural overview of the EJB programming environment

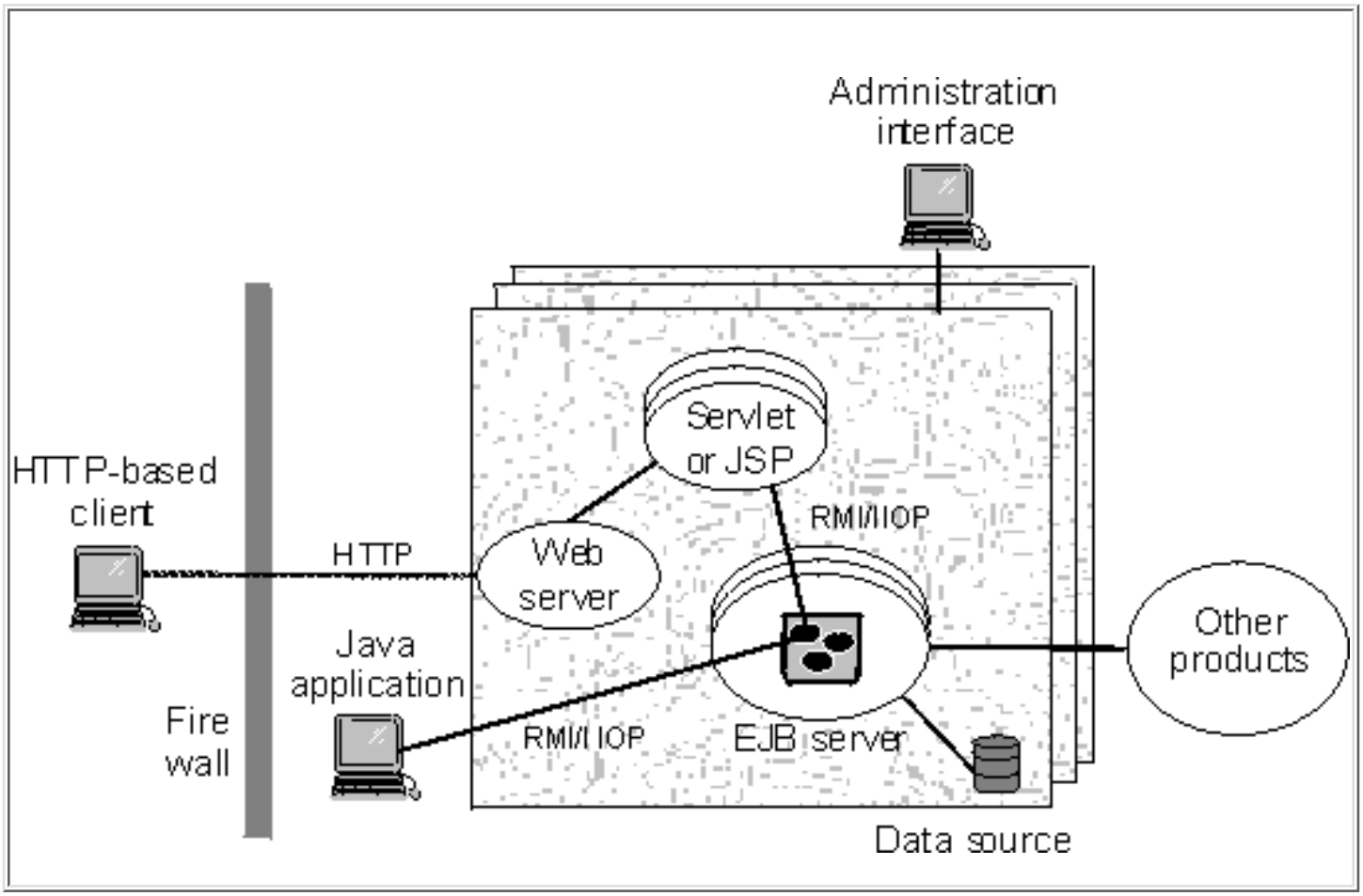
In the past few years, the World Wide Web (the Web) has transformed the wayin which businesses work with their customers. At first, it was goodenough just to have a Web home page. Then, businesses began to deployactive Web sites that allowed customers to order products and services.Today, businesses not only need to use the Web in all of these ways, they needto integrate their Web-based systems with their other business systems.The IBM(R) WebSphere Application Server, and specifically the support forenterprise beans, provides the model and the tools to accomplish thisintegration.

---

## Components of the EJB environment

IBM's implementation of the Sun Microsystems Enterprise JavaBeans (EJB)Specification enables users of the WebSphere Application Server AdvancedEdition and WebSphere Application Server Enterprise Edition to integrate theirWeb-based systems with their other business systems. A major part ofthis implementation is the WebSphere EJB server and its associated components,which are illustrated in Figure 1.

**Figure 1. The components of the EJB environment**




The WebSphere EJB server environment contains the following components,which are discussed in more detail in the specified sections:

- *EJB server*--A WebSphere EJB server contains and runs oneor more *enterprise beans*, which encapsulate the business logic anddata used and shared by EJB clients. The enterprise beans installed inan EJB server do not communicate directly with the server; instead, an*EJB container* provides an interface between the

enterprise beansand the EJB server, providing many low-level services such as threading,support for transactions, and management of data storage and retrieval.For more information, see The EJB server.

- *Data source*--There are two types of enterprisebeans: session beans, which encapsulate short-lived, client-specifictasks and objects, and entity beans, which encapsulate permanent or*persistent* data. The EJB server stores and retrieves thispersistent data in a data source, which can be a database, anotherapplication, or even a file. For more information, see The data source.
- *EJB clients*--There are two general types of EJBclients:
  - *HTTP-based clients* that interact with the EJB server by usingeither Java servlets or JavaServer Pages(TM) (JSP) by way of the HypertextTransfer Protocol (HTTP).
  - *Java applications* that interact directly with the EJB server byusing Java remote method invocation over the Internet Inter-ORB Protocol(RMI/IIOP).

  For more information, see The EJB clients.
- The *administration interface*--The administrative interfaceallows you to manage the EJB server environment. For more information,see The administration interface.

---

## The EJB server

The EJB server is the application server tier of WebSphere ApplicationServer's three-tier architecture, connecting the client tier (Javaservlets, applets, applications, and JSP) with the resource management tier(the data source). The WebSphere Application Server contains two typesof EJB servers. If you have the Advanced Application Server, you getonly one of these EJB servers; if you have the Enterprise ApplicationServer, you get both. When referring generically to EJB servers, thisdocumentation uses the phrase *EJB server*; when thedocumentation needs to refer specifically to one or the other, it uses thefollowing terms:

- *EJB server (AE)*--The EJB server that comes with theAdvanced Application Server. (Because Advanced Application Server isavailable as a part of Enterprise Application Server, this EJB server is alsoavailable with Enterprise Application Server.)
- *EJB server (CB)*--The EJB server that comes only with theEnterprise Application Server and is part of Component Broker (CB).

The EJB server has three components: the EJB server runtime, the EJBcontainers, and the enterprise beans. EJB containers insulate theenterprise beans from the underlying EJB server and provide a standardapplication programming interface (API) between the beans and thecontainer. The EJB Specification defines this API.

The EJB server (CB) includes two standard types of containers: entitycontainers and session containers. As their names imply, thesecontainers are specifically optimized to handle entity beans and sessionbeans, respectively. The EJB server (AE) has one standard containerthat supports both entity and session beans.Together, the EJB server and container components provide or give access tothe following services for the enterprise beans that are deployed intoit:

- A tool that deploys enterprise beans. When a bean is deployed, thedeployment tool creates several classes that implement the interfaces thatmake up the predeployed bean. In addition, the deployment toolgenerates Java ORB, stub, and skeleton classes that enable remote methodinvocation. For entity beans, the tool also generates persistor andfinder classes to handle interaction between the bean and the data source thatstores the bean's persistent data. Before an enterprise bean canbe deployed, the developer must create an *EJB module* and associated*deployment descriptor*. The deployment descriptor providesinformation about each enterprise bean in the module and instructions for thecontainer on how to handle the beans. For more information ondeployment, see Deploying an EJB module.
- A security service that handles authentication and authorization forprincipals that need to access resources in an EJB server environment.For more information, see The security service.
- A workload management service that ensures that resources are usedefficiently. For more information, see

The workload management service.

- A persistence service that handles interaction between an entity bean andits data source to ensure that persistent data is properly managed. Formore information, see The persistence service.
- A naming service that exports a bean's name, as defined in thedeployment descriptor, into the name space. The EJB server uses theJava Naming and Directory Interface(TM) (JNDI) to implement a namingservice. For more information, see The naming service.
- A transaction service that implements the transactional attributes in abean's deployment descriptor. For more information, see The transaction service.

## The security service

When enterprise computing was handled solely by a few powerful mainframeslocated at a centralized site, ensuring that only authorized users obtainedaccess to computing services and information was a fairly straightforwardtask. In distributed computing systems where users, applicationservers, and resource managers can be spread out across the world, securingcomputing resources has become a much more complicated task.Nevertheless, the underlying issues are basically the same.

#### Authentication and authorization

A good security service provides two main functions: authentication andauthorization.

*Authentication* takes place when a *principal* (a useror a computer process) initially attempts to gain access to a computingresource. At that point, the security service challenges the principalto prove that the principal is who it claims to be. Human userstypically prove who they are by entering a user ID and password; aprocess normally presents an encrypted key. If the password or key isvalid, the security service gives the user a *token* or*ticket* that identifies the principal and indicates that theprincipal has been authenticated.After a principal is authenticated, it can then attempt to use any of theresources within the boundaries of the computing system protected by thesecurity service; however, a principal can use a particular computingresource only if it has been authorized to do so.*Authorization* takes place when an authenticated principal requeststhe use of a resource and the security service determines if the user has beengranted permission to use that resource. Typically, authorization ishandled by associating access control lists (ACLs) with resources that definewhich principal (or groups of principals) are authorized to use theresource. If the principal is authorized, it gains access to theresource.

In a distributed computing environment, principals and resources must bemutually suspicious of each other's identity until both have proven thatthey are who they say they are. This is necessary because principalscan attempt to falsify an identity to get access to a resource, and a resourcecan be a trojan horse, attempting to get valuable information from theprincipal. To solve this problem, the security service contains asecurity server that acts as a *trusted third party*, authenticatingprincipals and resources so that these entities can prove their identities toeach other. This security protocol is known as *mutualauthentication*.

#### Using the security server in the EJB server environment

There are some similarities between the security service in the two EJBserver environments. In both EJB server environments, the securityservice does *not* use the *access control* and *run-asidentity* security attributes defined in the deploymentdescriptor. However, it does use the *run-as mode* attributeas the basis for mapping a user identity to a user security context.For more information on this attribute, see The deployment descriptor.

The major differences between the two security services are discussed inthe following sections.

Security in the EJB server (AE) environment

In the EJB server (AE) environment, the main component of the securityservice is an EJB server that contains security enterprise beans. Whensystem administrators administer the security service, they manipulate thesecurity beans in the security EJB server.

Once an EJB client is authenticated, it can attempt to invoke methods onthe enterprise beans that it manipulates. A method is successfullyinvoked if the principal associated with the method invocation has therequired permissions to invoke the method. These permissions can be setat the application level (an administrator-defined set of Web and objectresources) and at the method group level (an administrator-defined set of Javainterface/method pairs). An application can contain multiple methodgroups.

In general, the principal under which a method is invoked is associatedwith that invocation across multiple Web servers and EJB servers (thisassociation is known as *delegation*). Delegating the methodinvocations in this way ensures that the user of an EJB client needs toauthenticate only once. HTTP cookies are used to propagate auser's authentication information across multiple Web servers.These cookies have a lifetime equal to the life of the browser session, and alogout method is provided to destroy these cookies when the user isfinished.

For information on administering security in the EJB server (AE)environment, see the WebSphere InfoCenter and the online help available withthe WebSphere Administrative Console.

Security in the EJB server (CB) environmentIn the EJB server (CB) environment, you must secure all the Component Brokername servers and applications servers in the network. Securing the nameserver on each server host prevents unauthorized access to the system objects(including name contexts used in the Component Broker namespace) in thatserver. Securing an application server prevents unauthorized access tothe business objects for applications in that server.

To secure your name servers and application servers, you must do thefollowing:

- Install and configure the Distributed Computing Environment (DCE) toprovide authentication services to the servers. This allows secureaccess between servers.
- Configure key rings for clients and servers to provide authenticationservices to Java-based SSL clients.
- Configure authorization for access to business objects in the applicationservice.
- Create a delegation policy to allow the application server to pass therequesting client principal to other servers.
- Configure credential mapping to provide access to any third tiersystem.
- Configure the qualities of protection to be used to protect messages thatflow between clients and the application server.

The Component Broker System Administration Guideprovides more detail about each of these tasks.

## The workload management service

The workload management service improves the scalability of the EJB serverenvironment by grouping multiple EJB servers into *servergroups*. Clients then access these server groups as if they are asingle EJB server, and the workload management service ensures that theworkload is evenly distributed across the EJB servers in the servergroups. An EJB server can belong to only one server group.The creation of server groups is an administrative task that is handled fromwithin the WebSphere Administrative Console for the EJB server (AE)environment and from within the Systems Management End User Interface for theEJB server (CB) environment. For more information on workloadmanagement, consult the WebSphere InfoCenter and the online help for theappropriate administrative interface.

## The persistence service

There are two types of enterprise beans: session beans and entitybeans. Session beans encapsulate temporary data associated with aparticular client. Entity beans encapsulate permanent data that isstored in a data source. For more information, see An introduction to enterprise beans.

The persistence service ensures that the data associated with entity beansis properly synchronized with their corresponding data in the datasource. To accomplish this task, the persistence service works with thetransaction

service to insert, update, extract, and remove data from the datasource at the appropriate times.

There are two types of entity beans: those with container-managedpersistence (CMP) and those with bean-managed persistence (BMP). Inentity beans with CMP, the persistence service handles nearly all of the tasksrequired to manage persistent data. In entity beans with BMP, the beanitself handles most of the tasks required to manage persistent data.

In the EJB server (AE) environment, the persistence service uses thefollowing components to accomplish its task:

- The Java Database Connectivity (JDBC(TM)) API, which gives entity beans acommon interface to relational databases.
- Java transaction support, which is discussed in Using transactions in the EJB server environment. The EJB server ensures that persistent data isalways handled within the appropriate transactional context.

In the EJB server (CB) environment, the persistence service uses thefollowing components to accomplish its task:

- The X/Open XA interface, which gives entity beans a standard interface torelational databases.
- The Object Management Group's (OMG) Object Transaction Service (OTS),which is also discussed in Using transactions in the EJB server environment.

## The naming service

In an object-oriented distributed computing environment, clients must have amechanism to locate and identify objects so that the clients, objects, andresources appear to be on the same machine. A naming service providesthis mechanism. In the EJB server environment, JNDI is used to mask theactual naming service and provide a common interface to the namingservice.

JNDI provides naming and directory functionality to Java applications, butthe API is independent of any specific implementation of a naming anddirectory service. This implementation independence ensures thatdifferent naming and directory services can be used by accessing them by wayof the JNDI API. Therefore, Java applications can use many existingnaming and directory services such as the Lightweight Directory AccessProtocol (LDAP), the Domain Name Service (DNS), or the DCE Cell DirectoryService (CDS).

JNDI was designed for Java applications by using Java's objectmodel. Using JNDI, Java applications can store and retrieve namedobjects of any Java object type. JNDI also provides methods forexecuting standard directory operations, such as associating attributes withobjects and searching for objects by using their attributes.

In the EJB server environment, the deployment descriptor is used to specifythe JNDI name for an enterprise bean. When an EJB server is started, itregisters these names with JNDI.

## The transaction service

A *transaction* is a set of operations that transforms data from oneconsistent state to another. This set of operations is an indivisibleunit of work, and in some contexts, a transaction is referred to as a*logical unit of work* (LUW). A transaction is a tool fordistributed systems programming that simplifies failure scenarios. Transactions provide the *ACID properties*:

- *Atomicity*: A transaction's changes are atomic:either all operations that are part of the transaction happen or nonehappen.
- *Consistency*: A transaction moves data between consistentstates.
- *Isolation*: Even though transactions can run (or beexecuted) concurrently, no transaction sees another's work inprogress. The transactions appear to run serially.
- *Durability*: After a transaction completes successfully,its changes survive subsequent failures.

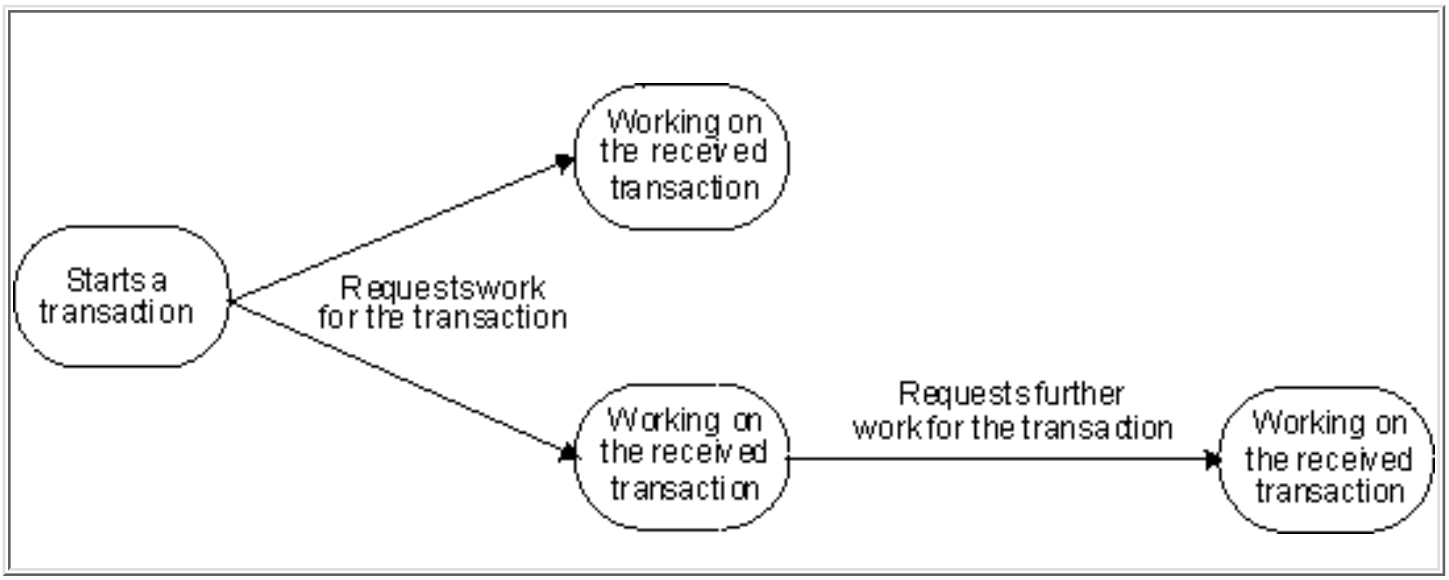
As an example, consider a transaction that transfers money from one accountto another. Such a transfer involves

money being deducted from oneaccount and deposited in the other. Withdrawing the money from oneaccount and depositing it in the other account are two parts of an*atomic* transaction: if both cannot be completed, neither musthappen. If multiple requests are processed against an account at thesame time, they must be *isolated* so that only a single transactioncan affect the account at one time. If the bank's central computerfails just after the transfer, the correct balance must still be shown whenthe system becomes available again: the change must be*durable*. Note that *consistency* is a function ofthe application; if money is to be transferred from one account toanother, the application must subtract the same amount of money from oneaccount that it adds to the other account. Transactions can be completed in one of two ways: they can commit orroll back. A successful transaction is said to*commit*. An unsuccessful transaction is said to *rollback*. Any data modifications made by a rolled back transactionmust be completely undone. In the money-transfer example, if money iswithdrawn from one account but a failure prevents the money from beingdeposited in the other account, any changes made to the first account must becompletely undone. The next time any source queries the accountbalance, the correct balance must be shown.

#### Distributed transactions and the two-phase commit process

A *distributed transaction* is one that runs in multiple processes,often on several machines. Each process participates in thetransaction. This is illustrated in Figure 2, where each oval indicates work being done on adifferent machine, and each arrow indicates a remote method invocation(RMI).

**Figure 2. Example of a distributed transaction**




Distributed transactions, like local transactions, must adhere to the ACIDproperties. However, maintaining these properties is greatlycomplicated for distributed transactions because a failure can occur in anyprocess, and in the event of such a failure, each process must undo any workalready done on behalf of the transaction.

A distributed transaction processing system maintains the ACID propertiesin distributed transactions by using two features:

- *Recoverable processes*: Recoverable processes are thosethat can restore earlier states if a failure occurs.
- *A commit protocol*: A commit protocol enables multipleprocesses to coordinate the committing or rolling back (aborting) of atransaction. The most common commit protocol, and the one used by theEJB server, is the two-phase commit protocol.

Transaction state information must be stored by all recoverableprocesses. However, only processes that manage application data (suchas resource managers) must store descriptions of changes to data. Notall processes involved in a distributed transaction need to berecoverable. In general, clients are not recoverable because they donot interact directly with a resource manager. Processes that are notrecoverable are referred to as *ephemeral* processes.The *two-phase commit protocol*, as the name implies, involves twophases: a prepare phase and a resolution phase. In eachtransaction, one process acts as the coordinator. The*coordinator* oversees the activities

of the other participants inthe transaction to ensure a consistent outcome. In the *prepare phase*, the coordinator sends a message to eachprocess in the transaction, asking each process to prepare to commit.When a process prepares, it guarantees that it can commit the transaction andmakes a permanent record of its work. After guaranteeing that it cancommit, it can no longer unilaterally decide to roll back thetransaction. If a process cannot prepare (that is, if it cannotguarantee that it can commit the transaction), it must roll back thetransaction.In the *resolution phase*, the coordinator tallies theresponses. If all participants are prepared to commit, the transactioncommits; otherwise, the transaction is rolled back. In eithercase, the coordinator informs all participants of the result. In thecase of a commit, the participants acknowledge that they havecommitted.

#### Using transactions in the EJB server environment

The enterprise bean transaction model corresponds in most respects to theOMG OTS version 1.1. An enterprise bean instance that istransaction enabled corresponds to an object of the OTS TransactionalObjectinterface. However, the enterprise bean transaction model does notsupport transaction nesting.

In the EJB server environment, transactions are handled by three maincomponents of the transaction service:

- A transaction manager interface that enables the EJB server to controltransaction boundaries within its enterprise beans based on the transactionalattributes specified for the beans.
- An interface (UserTransaction) that allows an enterprise bean or an EJBclient to manage transactions. The container makes this interfaceavailable to enterprise beans and EJB clients by way of the nameservice.
- Coordination by way of the X/Open XA interface that enables atransactional resource manager (such as a database) to participate in atransaction controlled by an external transaction manager.

For most purposes, the enterprise bean developers can delegate the tasksinvolved in managing a transaction to the container. The developerperforms this delegation by setting the deployment descriptor attributes fortransactions. These attributes and their values are described in Setting transactional attributes in the deployment descriptor.

In other cases, the enterprise bean developer will want or need to managethe transactions at the bean level or involve the EJB client in the managementof transactions. For more information on this approach, see Using bean-managed transactions.

---

## The data source

Entity beans contain persistent data that must be permanently stored in arecoverable data source. Although the EJB Specification often refers todatabases as the place to store persistent data associated with an entitybean, it leaves open the possibility of using other data sources, includingoperating system files and other applications.If you want to let the container handle the interaction between an entity beanand a data source, you must use the data sources supported by thatcontainer:

- The EJB server (AE) supports DB2(R), Oracle, Sybase, andInstantDB.
- The EJB server (CB) supports DB2, Oracle, CICS(R), IMS(TM), andMQSeries(R).

If you write the additional code required to handle the interaction betweena BMP entity bean and the data source, you can use any data source that meetsyour needs and is compatible with the persistence service. For moreinformation, see Developing entity beans with BMP.

---

## The EJB clients

An EJB client can take one of the following forms: it can be a Javaapplication, a Java servlet, a Java applet-servlet combination, or a JSPfile. For the EJB server (CB), a Java applet can be used to directlyinteract with enterprise beans. For the EJB server (AE), a Java appletcan be used only in combination with a servlet.

The EJB client code required to access and manipulate enterprise beans isvery similar across the different Java EJB clients. EJB clientdevelopers must consider the following issues:

- *Naming and communications*--A Java EJB client must useeither HTTP or RMI to communicate with enterprise beans. Fortunately,there is very little difference in the coding required to enablecommunications between the EJB client and the enterprise bean, because JNDImasks the interaction between the EJB client and the name service.
    - Java applications communicate with enterprise beans by usingRMI/IIOP.
    - Java servlets and JSP files communicate with enterprise beans by usingHTTP. To use servlets with an EJB server, a Web server must beinstalled and configured on a machine in the EJB server environment.For more information, see The Web server.
- *Threading*--Java clients can be either single-threaded ormultithreaded depending on the tasks that the client needs to perform.Each client thread that uses a service provided by a session bean must createor find a separate instance of that bean and maintain a reference to that beanuntil the thread completes; multiple client threads can access the sameentity bean.
- *Security*
    - EJB clients that access an EJB server (AE) over HTTP (for example,servlets and JSP files) encounter the following two layers of security:
        1. Universal Resource Locator (URL) security enforced by the WebSphereApplication Server Security Plug-in attached to the Web server incollaboration with the security service.
        2. Enterprise bean security enforced at the server working with the securityservice.

      When the user of an HTTP-based EJB client attempts to access an enterprisebean, the Web server (using the WebSphere Server plug-in) authenticates theuser. This authentication can take the form of a request for a user IDand password or it can happen transparently in the form of a certificateexchange followed by the establishment of a Secure Sockets Layer (SSL)session.

      The authentication policy is governed by an additional option: securechannel constraint. If the secure channel constraint is required, anSSL session must be established as the final phase of authentication;otherwise, SSL is optional.
    - All EJB clients that access an EJB server (CB) and EJB clients that accessan EJB server (AE) by using RMI (for example, Java applications) encounter thesecond security layer only. Like HTTP-based EJB clients, these EJBclients must authenticate with the security service.

      For more information, see The security service.
- *Transactions*--Both types of Java clients can use thetransaction service by way of the JTA interfaces to managetransactions. The code required for transaction management is identicalin the two types of clients. For general information on transactionsand the Java transaction service, see The transaction service. For information on managing transactions in a JavaEJB client, see Managing transactions in an EJB client.

In the EJB server (CB) environment, an enterprise bean can also be accessedby EJB clients that use Microsoft(R) ActiveX(R), CORBA-based Java, andto a limited degree, CORBA-based C++. More information on EJB clients specific to the EJB server (CB) provides additional information.

**Note:**

In the EJB server (AE) environment, ActiveX and CORBA-based access toenterprise beans is not supported.

---

## The Web server

To access the functionality in the EJB server, Java servlets and JSP filesmust have access to a Web server. The

Web server enables communicationbetween a Web client and the EJB server. The EJB server, Web server,and Java servlet can each reside on different machines.

For information on the Web servers supported by the EJB servers, see theAdvanced Application Server *Getting Started* document.

---

## The administration interface

The EJB server (CB) and EJB server (AE) each have their own administrationtools:

- The EJB server (AE) uses the WebSphere Administrative Console. Formore information on this interface, consult the WebSphere InfoCenter and theonline help available with the WebSphere Administrative Console.
- The EJB server (CB) uses the System Management End User Interface (SMEUI). For more information on this interface, see the Component BrokerSystem Administration Guide.

You can also administer the EJB server (AE) using the **wscp**command-line tool. For more information, see the Advanced EditionInformation Center.

# More-advanced programming concepts for enterprise beans

This chapter discusses some of the more advanced programming conceptsassociated with developing and using enterprise beans. It includesinformation on developing entity beans with bean-managed persistence (BMP),writing the code required by a BMP bean to interact with a database, anddeveloping session beans that directly participate in transactions.

---

## Developing entity beans with BMP

In an entity bean with container-managed persistence (CMP), the containerhandles the interactions between the enterprise bean and the datasource. In an entity bean with bean-managed persistence (BMP), theenterprise bean must contain all of the code required for the interactionsbetween the enterprise bean and the data source. For this reason,developing an entity bean with CMP is simpler than developing an entity beanwith BMP. However, you must use BMP if any of the following is trueabout an entity bean:

- The bean's persistent data is stored in more than one datasource.
- The bean's persistent data is stored in a data source that is notsupported by the EJB server that you are using.

This section examines the development of entity beans with BMP. Forinformation on the tasks required to develop an entity bean with CMP, see Developing entity beans with CMP.

Every entity bean must contain the following basic parts:

- The enterprise bean class. For more information, see Writing the enterprise bean class (entity with BMP).
- The enterprise bean's home interface. For more information,see Writing the home interface (entity with BMP).
- The enterprise bean's remote interface. For more information,see Writing the remote interface (entity with BMP).

In an entity bean with BMP, you can create your own primary key class oruse an existing class for the primary key. For more information, see Writing or selecting the primary key class (entity with BMP).

### Writing the enterprise bean class (entity with BMP)

In an entity bean with BMP, the bean class defines and implements the businessmethods of the enterprise bean, defines and implements the methods used tocreate instances of the enterprise bean, and implements the methods invoked bythe container to move the bean through different stages in the bean'slife cycle.

By convention, the enterprise bean class is named *Name*Bean,where *Name* is the name you assign to the enterprise bean.The enterprise bean class for the example AccountBM enterprise bean is namedAccountBMBean.Every entity bean class with BMP must meet the following requirements:

- It must be public, it must *not* be abstract, and it mustimplement the javax.ejb.EntityBean interface. For moreinformation, see Implementing the EntityBean interface.
- It must define instance variables that correspond to persistent dataassociated with the enterprise bean. For more information, see Defining instance variables.
- It must implement the business methods used to access and manipulate thedata associated with the enterprise bean. For more information, see Implementing the business methods.
- It must contain code for getting connections to, interacting with, andreleasing connections to the data source (or sources) used to store thepersistent data. For more information, see Using a database with a BMP entity bean.
- It must define and implement an ejbCreate method for each way in which theenterprise bean can be instantiated. It can, but is not required to,define and implement a corresponding ejbPostCreate method for each ejbCreatemethod. For more information, see Implementing the ejbCreate and ejbPostCreate methods.
- It must implement the ejbFindByPrimaryKey method that takes a primary keyand determines if it is valid and unique. It can also define andimplement additional finder methods as required. For more information,see Implementing the ejbFindByPrimaryKey and other ejbFind methods.

**Note:**

The enterprise bean class can implement the enterprise bean's remoteinterface, but this is not recommended. If the enterprise bean classimplements the remote interface, it is possible to inadvertently pass the*this* variable as a method argument.

Figure 55 shows the import statements and class declaration for theexample AccountBM enterprise bean.

**Figure 55. Code example: The AccountBMBean class**

```
...import java.rmi.RemoteException;import java.util.*;import javax.ejb.*;import java.lang.*;import
java.sql.*;import com.ibm.ejs.doc.account.InsufficientFundsException;public class AccountBMBean
implements EntityBean {   ...}
```

#### Defining instance variables

An entity bean class can contain both persistent and nonpersistent instancevariables; however, static variables are not supported in enterprisebeans

unless they are also final (that is, they are constants).Persistent variables are stored in a database. Unlike the persistentvariables in a CMP entity bean class, the persistent variables in a BMP entitybean class can be private.

Nonpersistent variables are *not* stored in a database and aretemporary. Nonpersistent variables must be used with caution and mustnot be used to maintain the state of an EJB client between methodinvocations. This restriction is necessary because nonpersistentvariables cannot be relied on to remain the same between method invocationsoutside of a transaction because other EJB clients can change these variablesor they can be lost when the entity bean is passivated.

The AccountBMBean class contains three instance variables that representpersistent data associated with the AccountBM enterprise bean:

- *accountId*, which identifies the account ID associated with anaccount
- *type*, which identifies the account type as either savings (1)or checking (2)
- *balance*, which identifies the current balance of the account

The AccountBMBean class contains several nonpersistent instance variablesincluding the following:

- *entityContext*, which identifies the entity context of eachinstance of an AccountBM enterprise bean. The entity context can beused to get a reference to the EJB object currently associated with the beaninstance and to get the primary key object associated with that EJBobject.
- *jdbcUrl*, which encapsulates the database universal resourcelocator (URL) used to connect to the data source. This variable musthave the following format:*dbAPI*:*databaseType*:*databaseName*.For example, to specify a database named sample in an IBM DB2 database withthe Java Database Connectivity (JDBC) API, the argument is`jdbc:db2:sample`.
- *driverName*, which encapsulates the database driver classrequired to connect to the database.
- *DBLogin*, which identifies the database user ID required toconnect to the database.
- *DBPassword*, which identifies password for the specified user ID(*DBLogin*) required to connect to the database.
- *tableName*, which identifies the database table name in whichthe bean's persistent data is stored.
- *jdbcConn*, which encapsulates a Java Database Connectivity(JDBC) connection to a data source within a java.sql.Connectionobject.

**Figure 56. Code example: The instance variables of the AccountBMBean class**

```
...public class AccountBMBean implements EntityBean {      private EntityContext entityContext =
null;     ...     private static final String DBRULProp = "DBURL";     private static final String
DriverNameProp = "DriverName";     private static final String DBLoginProp = "DBLogin";     private
static final String DBPasswordProp = "DBPassword";     private static final String TableNameProp =
"TableName";     private String jdbcUrl, driverName, DBLogin, DBPassword, tableName;     private
long accountId = 0;     private int type = 1;     private float balance = 0.0f;          private
Connection jdbcConn = null;      ...}
```

To make the AccountBM bean more portable between databases and databasedrivers, the database-specific variables (*jdbcUrl*,*driverName*, *DBLogin*, *DBPassword*, and*tableName*) are set by retrieving corresponding environmentvariables contained in the enterprise bean. The values of thesevariables are retrieved by the getEnvProps method, which is implemented in theAccountBMBean class and invoked when the setEntityContext method iscalled. For more information, see Managing connections in the EJB server (CB) environment or Managing database connections in the EJB server (AE) environment.

For more information on how to set an enterprise bean's environmentvariables, refer to Setting environment variables for an enterprise bean.

Although Figure 56 shows database access compatible with version 1.0 ofthe JDBC specification, you can also perform database accesses that arecompatible with version 2.0 of the JDBC specification. Anadministrator binds a javax.sql.DataSource reference (whichencapsulates the information that was formerly stored in the jdbcURL anddriverName variables) into the JNDI namespace. The entity bean with BMPdoes the following to get a java.sql.Connection:

```
DataSource ds = (dataSource)initialContext.lookup("java:comp/env/jdbc/MyDataSource");Connection con
= ds.getConnection();
```

where *MyDataSource* is the name the administrator assigned to thedatasource.

#### Implementing the business methods

The business methods of an entity bean class define the ways in which the dataencapsulated in the class can be manipulated. The business methodsimplemented in the enterprise bean class cannot be directly invoked by an EJBclient. Instead, the EJB client invokes the corresponding methodsdefined in the enterprise bean's remote interface by using an EJB objectassociated with an instance of the enterprise bean, and the container invokesthe corresponding methods in the instance of the enterprise bean.

Therefore, for every business method implemented in the enterprise beanclass, a corresponding method must be defined in the enterprise bean'sremote interface. The enterprise bean's remote interface isimplemented by the container in the EJB object class when the enterprise beanis deployed.

There is no difference between the business methods defined in theAccountBMBean bean class and those defined in the CMP bean class AccountBeanshown in Figure 20.

#### Implementing the ejbCreate and ejbPostCreate methods

You must define and implement an ejbCreate method for each way in which youwant a new instance of an enterprise bean to be created. For eachejbCreate method, you can also define a corresponding ejbPostCreatemethod. Each ejbCreate method must correspond to a create method in

theEJB home interface.

Like the business methods of the bean class, the ejbCreate andejbPostCreate methods cannot be invoked directly by the client.Instead, the client invokes the create method of the enterprise bean'shome interface by using the EJB home object, and the container invokes theejbCreate method followed by the ejbPostCreate method.

Unlike the method in an entity bean with CMP, the ejbCreate method in anentity bean with BMP must contain all of the code required to insert thebean's persistent data into the data source. This requirementmeans that the ejbCreate method must get a connection to the data source (ifone is not already available to the bean instance) and insert the values ofthe bean's variables into the appropriate fields in the datasource.

Each ejbCreate method in an entity bean with BMP must meet the followingrequirements:

- It must be public and return the bean's primary key class.
- Its arguments and return type must be valid for Java remote methodinvocation (RMI).
- It must contain the code required to insert the values of the persistentvariables into the data source. For more information, see Using a database with a BMP entity bean.

Each ejbPostCreate method must be public, return void, and have the samearguments as the matching ejbCreate method.If necessary, both the ejbCreate method and the ejbPostCreate method can throwthe java.rmi.RemoteException exception, thejavax.ejb.CreateException exception, thejavax.ejb.DuplicateKeyException exception, and any user-definedexceptions.

Figure 57 shows the two ejbCreate methods required by the exampleAccountBMBean bean class. No ejbPostCreate methods are required.

As in the AccountBean class, the first ejbCreate method calls the secondejbCreate method; the latter handles all of the interaction with the datasource. The second method initializes the bean's instancevariables and then ensures that it has a valid connection to the data sourceby invoking the checkConnection method. The method then creates,prepares, and executes an SQL INSERT call on the data source. If theINSERT call is executed correctly, and only one row is inserted into the datasource, the method returns an object of the bean's primary keyclass.

**Figure 57. Code example: The ejbCreate methods of the AccountBMBean class**

```
public AccountBMKey ejbCreate(AccountBMKey key) throws CreateException,      RemoteException {
return ejbCreate(key, 1, 0.0f); }...public AccountBMKey ejbCreate(AccountBMKey key, int type, float
balance)     throws CreateException, RemoteException{     accountId = key.accountId;     this.type =
type;     this.balance = balance;     checkConnection();     // INSERT into database     try {
String sqlString = "INSERT INTO " + tableName +               " (balance, type, accountid) VALUES
(?,?,?)";          PreparedStatement sqlStatement = jdbcConn.prepareStatement(sqlString);
sqlStatement.setFloat(1, balance);          sqlStatement.setInt(2, type);
sqlStatement.setLong(3, accountId);          // Execute query          int updateResults =
sqlStatement.executeUpdate();          ...     }     catch (Exception e) { // Error occurred during
insert          ...     }     return key;}
```

#### Implementing the ejbFindByPrimaryKey and other ejbFind methods

At a minimum, each entity bean with BMP must define and implement theejbFindByPrimaryKey method that takes a primary key and determines if it isvalid and unique for an instance of an enterprise bean; if the primarykey is valid and unique, it returns the primary key. An entity bean canalso define and implement other finder methods to find enterprise beaninstances. All finder methods can throw thejavax.ejb.FinderException exception to indicate anapplication-level error. Finder methods designed to find a single beancan also throw the javax.ejb.ObjectNotFoundException exception,a subclass of the FinderException class. Finder methods designed toreturn multiple beans should not use the ObjectNotFoundException to indicatethat no suitable beans were found; instead, such methods should returnempty return values. Throwing thejava.rmi.RemoteException exception is deprecated; see Standard application exceptions for entity beans for more information.

Like the business methods of the bean class, the ejbFind methods cannot beinvoked directly by the client. Instead, the client invokes a findermethod on the enterprise bean's home interface by using the EJB homeobject, and the container invokes the corresponding ejbFind method. Thecontainer invokes an ejbFind method by using a generic instance of that entitybean in the pooled state.

Because the container uses an instance of an entity bean in the pooledstate to invoke an ejbFind method, the method must do the following:

1. Get a connection to the data source (or sources).
2. Query the data source for records that match specifications of the findermethod.
3. Drop the connection to the data source (or sources).

For more information on these data source tasks, see Using a database with a BMP entity bean.Figure 58 shows the ejbFindByPrimaryKey method of the exampleAccountBMBean class. The ejbFindByPrimaryKey method gets a connectionto its data source by calling the makeConnection method shown in Figure 58. It then creates and invokes an SQL SELECT statementon the data source by using the specified primary key.

If one and only one record is found, the method returns the primary keypassed to it in the argument. If no records are found or multiplerecords are found, the method throws the FinderException. Beforedetermining whether to return the primary key or throw the FinderException,the method drops its connection to the data source by calling thedropConnection method described in Using a database with a BMP entity bean.

**Figure 58. Code example: The ejbFindByPrimaryKey method of the AccountBMBean class**

```
public AccountBMKey ejbFindByPrimaryKey (AccountBMKey key)    throws FinderException {       boolean
wasFound = false;       boolean foundMultiples = false;       makeConnection();       try {
// SELECT from database            String sqlString = "SELECT balance, type, accountid FROM " +
tableName                  + " WHERE accountid = ?";            PreparedStatement sqlStatement =
jdbcConn.prepareStatement(sqlString);            long keyValue = key.accountId;
sqlStatement.setLong(1, keyValue);                        // Execute query            ResultSet
sqlResults = sqlStatement.executeQuery();                        // Advance cursor (there should be
only one item)            // wasFound will be true if there is one           wasFound =
sqlResults.next();                       // foundMultiples will be true if more than one is found.
foundMultiples = sqlResults.next();       }        catch (Exception e)  { // DB error           ...
}       dropConnection();       if (wasFound && !foundMultiples)       {           return key;
}        else       {           // Report finding no key or multiple keys           ...
throw(new FinderException(foundStatus));      }}
```

Figure 59 shows the ejbFindLargeAccounts method of the exampleAccountBMBean class. The ejbFindLargeAccounts method also gets aconnection to its data source by calling the makeConnection method and dropsthe connection by using the dropConnection method. The SQL SELECTstatement is also very similar to that used by the ejbFindByPrimaryKeymethod. (For more information on these data source tasks and methods,see Using a database with a BMP entity bean.)

While the ejbFindByPrimaryKey method needs to return only one primary key,the ejbFindLargeAccounts method can be expected to return zero or more primarykeys in an Enumeration object. To return an enumeration of primarykeys, the ejbFindLargeAccounts method does the following:

1. It uses a while loop to examine the result set (*sqlResults*)returned by the executeQuery method.
2. It inserts each primary key in the result set into a hash table named*resultTable* by wrapping the returned account ID in a Long objectand then in an AccountBMKey object. (The Long object,*memberId*, is used as the hash table's index.)
3. It invokes the elements method on the hash table to obtain the enumerationof primary keys, which it then returns.

**Figure 59. Code example: The ejbFindLargeAccounts method of the AccountBMBean class**

```
public Enumeration ejbFindLargeAccounts(float amount) throws FinderException {     makeConnection();
Enumeration result;     try  {          // SELECT from database          String sqlString = "SELECT
accountid FROM " + tableName                + " WHERE balance >= ?";          PreparedStatement
sqlStatement = jdbcConn.prepareStatement(sqlString);          sqlStatement.setFloat(1, amount);
// Execute query          ResultSet sqlResults = sqlStatement.executeQuery();          // Set up
Hashtable to contain list of primary keys          Hashtable resultTable = new Hashtable();
// Loop through result set until there are no more entries          // Insert each primary key into
the resultTable          while(sqlResults.next() == true)  {              long acctId =
sqlResults.getLong(1);              Long memberId = new Long(acctId);              AccountBMKey
key = new AccountBMKey(acctId);              resultTable.put(memberId, key);          }          //
Return the resultTable as an Enumeration          result = resultTable.elements();          return
result;     } catch (Exception e)  {          ...     } finally {          dropConnection();     }}
```

#### Implementing the EntityBean interface

Each entity bean class must implement the methods inherited from thejavax.ejb.EntityBean interface. The container invokesthese methods to move the bean through different stages in the bean'slife cycle. Unlike an entity bean with CMP, in an entity bean with BMP,these methods must contain all of the code for the required interaction withthe data source (or sources) used by the bean to store its persistentdata.

- ejbActivate--This method is invoked by the container when thecontainer selects an entity bean instance from the instance pool and assignsthat instance to a specific existing EJB object. This method mustcontain the code required to activate the enterprise bean instance by gettinga connection to the data source and using the bean'sjavax.ejb.EntityContext class to obtain the primary key in thecorresponding EJB object.

  In the example AccountBMBean class, the ejbActivate method obtains the beaninstance's account ID, sets the value of the *accountId*variable, and invokes the checkConnection method to ensure that it has a validconnection to the data source.

- ejbLoad--This method is invoked by the container to synchronize anentity bean's persistent variables with the corresponding data in thedata source. (That is, the values of the fields in the data source areloaded into the persistent variables in the corresponding enterprise beaninstance.) This method must contain the code required to load thevalues from the data source and assign those values to the bean'sinstance variables.

  In the example AccountBMBean class, the ejbLoad method obtains the beaninstance's account ID, sets the value of the *accountId*variable, invokes the checkConnection method to ensure that it has a validconnection to the data source, constructs and executes an SQL SELECTstatement, and sets the values of the *type* and *balance*variables to match the values retrieved from the data source.

- ejbPassivate--This method is invoked by the container to disassociatean entity bean instance from its EJB object and place the enterprise beaninstance in the instance pool. This method must contain the coderequired to "passivate" or deactivate an enterprise bean instance.Usually, this passivation simply means dropping the connection to the datasource.

  In the example AccountBMBean class, the ejbPassivate method invokes thedropConnection method to drop the connection to the data source.

- ejbRemove--This method is invoked by the container when a clientinvokes the remove method inherited by the enterprise bean's homeinterface (from the javax.ejb.EJBHome interface) or remoteinterface (from the javax.ejb.EJBObject interface). Thismethod must contain the code required to remove an enterprise bean'spersistent data from the data source. This method can throw thejavax.ejb.RemoveException exception if removal of an enterprisebean instance is not permitted. Usually, removal involves deleting thebean instance's data from the data source and

then dropping the beaninstance's connection to the data source.

In the example AccountBMBean class, the ejbRemove method invokes thecheckConnection method to ensure that it has a valid connection to the datasource, constructs and executes an SQL DELETE statement, and invokes thedropConnection method to drop the connection to the data source.

- setEntityContext--This method is invoked by the container to pass areference to the javax.ejb.EntityContext interface to anenterprise bean instance. This method must contain any code required tostore a reference to a context.

  In the example AccountBMBean class, the setEntityContext method sets thevalue of the *entityContext* variable to the value passed to it bythe container.
- ejbStore--This method is invoked by the container when the containerneeds to synchronize the data in the data source with the values of thepersistent variables in an enterprise bean instance. (That is, thevalues of the variables in the enterprise bean instance are copied to the datasource, overwriting the previous values.) This method must contain thecode required to overwrite the data in the data source with the correspondingvalues in the enterprise bean instance.

  In the example AccountBMBean class, the ejbStore method invokes thecheckConnection method to ensure that it has a valid connection to the datasource and constructs and executes an SQL UPDATE statement.
- unsetEntityContext--This method is invoked by the container, beforean enterprise bean instance is removed, to free up any resources associatedwith the enterprise bean instance. This is the last method called priorto removing an enterprise bean instance.

  In the example AccountBMBean class, the unsetEntityContext method sets thevalue of the *entityContext* variable to null.

## Writing the home interface (entity with BMP)

An entity bean's home interface defines the methods used by EJB clientsto create new instances of the bean, find and remove existing instances, andobtain metadata about an instance. The home interface is defined by theenterprise bean developer and implemented in the EJB home class created by thecontainer during enterprise bean deployment. The container makes thehome interface accessible to clients through the Java Naming and DirectoryInterface (JNDI).

By convention, the home interface is named *Name*Home, where*Name* is the name you assign to the enterprise bean. Forexample, the AccountBM enterprise bean's home interface is namedAccountBMHome.Every home interface for an entity bean with BMP must meet the followingrequirements:

- It must extend the javax.ejb.EJBHome interface. Thehome interface inherits several methods from thejavax.ejb.EJBHome interface. See The javax.ejb.EJBHome interface for information on thesemethods.
- Each method in the interface must be either a create method, whichcorresponds to an ejbCreate method (and possibly an ejbPostCreate method) inthe enterprise bean class, or a finder method, which corresponds to an ejbFindmethod in the enterprise bean class. For more information, see Defining create methods and Defining finder methods.
- The parameters and return value of each method defined in the homeinterface must be valid for Java RMI. For more information, see The java.io.Serializable and java.rmi.Remote interfaces. In addition, each method's throws clause mustinclude the java.rmi.RemoteException exception class.

Figure 60 shows the relevant parts of the definition of the homeinterface (AccountBMHome) for the example AccountBM bean. Thisinterface defines two abstract create methods: the first creates anAccountBM object by using an associated AccountBMKey object, the secondcreates an AccountBM object by using an associated AccountBMKey object andspecifying an account type and an initial balance. The interfacedefines the required findByPrimaryKey method and the findLargeAccountsmethod.

**Figure 60. Code example: The AccountBMHome home interface**

```
...import java.rmi.*;import javax.ejb.*;import java.util.*;public interface  AccountBMHome extends
EJBHome  {      ...     AccountBM create(AccountBMKey key) throws CreateException,
RemoteException;     ...     AccountBM create(AccountBMKey key, int type, float amount)
throws CreateException, RemoteException;     ...     AccountBM findByPrimaryKey(AccountBMKey key)
throws FinderException, RemoteException;     ...     Enumeration findLargeAccounts(float amount)
throws FinderException, RemoteException;}
```

#### Defining create methods

A create method is used by a client to create an enterprise bean instance andinsert the data associated with that instance into the data source.Each create method must be named create and it must have the same number andtypes of arguments as a corresponding ejbCreate method in the enterprise beanclass. (The ejbCreate method can itself have a correspondingejbPostCreate method.) The return types of the create method and itscorresponding ejbCreate method are always different.

Each create method must meet the following requirements:

- It must be named create.
- It must return the type of the enterprise bean's remoteinterface. For example, the return type for the create methods in theAccountBMHome interface is AccountBM (as shown in Figure 23).
- It must have a throws clause that includes thejava.rmi.RemoteException exception, thejavax.ejb.CreateException exception, and all of the exceptionsdefined in the throws clause of the corresponding ejbCreate and ejbPostCreatemethods.

#### Defining finder methods

A finder method is used to find one or more existing entity EJBobjects. Each finder method must be named find*Name*, where*Name* further describes the finder method's purpose.

At a minimum, each home interface must define the findByPrimaryKey methodthat enables a client to locate an EJB object by using the primary keyonly. The findByPrimaryKey method has one argument, an object of thebean's primary key class, and returns the type of the bean's remoteinterface.

Every other finder method must meet the following requirements:

- It must return the type of the enterprise bean's remote interface,the java.util.Enumeration interface, or thejava.util.Collection interface (when a finder method can returnmore than one EJB object or an EJB collection).
- It must have a throws clause that includes thejava.rmi.RemoteException andjavax.ejb.FinderException exception classes.

Although every entity bean must contain only the default finder method, youcan write additional ones if needed. For example, the AccountBMbean's home interface defines the findLargeAccounts method to findobjects that encapsulate accounts with balances of more than a specifieddollar amount, as shown in Figure 60. Because this finder method can be expected to returna reference to more than one EJB object, its return type isjava.util.Enumeration.

Unlike the implementation in an entity bean with CMP, in an entity beanwith BMP, the bean developer must fully implement the ejbFindByPrimaryKeymethod that corresponds to the findByPrimaryKey method. In addition,the bean developer must write each additional ejbFind method corresponding tothe finder methods defined in the home interface. The implementation ofthe ejbFind methods in the AccountBMBean class is discussed in Implementing the ejbFindByPrimaryKey and other ejbFind methods.

### Writing the remote interface (entity with BMP)

An entity bean's remote interface provides access to the business methodsavailable in the bean class. It also provides methods to remove an EJBobject associated with a bean instance and to obtain the bean instance'shome interface, object handle, and primary key. The remote interface isdefined by the EJB developer and implemented in the EJB object class createdby the container during enterprise bean deployment.

By convention, the remote interface is named *Name*, where*Name* is the name you assign to the enterprise bean. Forexample, the AccountBM enterprise bean's remote interface is namedAccountBM.Every remote interface must meet the following requirements:

- It must extend the javax.ejb.EJBObject interface. Theremote interface inherits several methods from thejavax.ejb.EJBObject interface. See Methods inherited from javax.ejb.EJBObject for information on thesemethods.
- It must define a corresponding business method for every business methodimplemented in the enterprise bean class.
- The parameters and return value of each method defined in the interfacemust be valid for Java RMI. For more information, see The java.io.Serializable and java.rmi.Remote interfaces.
- Each method's throws clause must include thejava.rmi.RemoteException exception class.

Figure 61 shows the relevant parts of the definition of the remoteinterface (AccountBM) for the example AccountBM enterprise bean. Thisinterface defines four methods for displaying and manipulating the accountbalance that exactly match the business methods implemented in theAccountBMBean class.All of the business methods throw the java.rmi.RemoteExceptionexception class. In addition, the subtract method must throw theuser-defined exceptioncom.ibm.ejs.doc.account.InsufficientFundsExceptionbecause the corresponding method in the bean class throws thisexception. Furthermore, any client that calls this method must eitherhandle the exception or pass it on by throwing it.

**Figure 61. Code example: The AccountBM remote interface**

```
...import java.rmi.*;import javax.ejb.*;import
com.ibm.ejs.doc.account.InsufficientFundsException;public interface AccountBM extends EJBObject {
...     float add(float amount) throws RemoteException;     ...     float getBalance() throws
RemoteException;     ...     void setBalance(float amount) throws RemoteException;     ...     float
subtract(float amount) throws InsufficientFundsException,          RemoteException;}
```

### Writing or selecting the primary key class (entity with BMP)

Every entity EJB object has a unique identity within a container that isdefined by a combination of the object's home interface name and itsprimary key, the latter of which is assigned to the object at creation.If two EJB objects have the same identity, they are consideredidentical.

The primary key class is used to encapsulate an EJB object's primarykey. In an entity bean (with BMP or CMP), you can write a distinctprimary key class or you can use an existing class as the primary key class,as long as that class is serializable. For more information, see The java.io.Serializable and java.rmi.Remote interfaces.

The example AccountBM bean uses a primary key class that is identical tothe AccountKey class contained in the Account bean shown in Figure 26, with the exception that the key class is namedAccountBMKey.

**Note:**

For the EJB server (AE) environment, the primary key class of an entity beanwith BMP must implement the hashCode and equals method. In addition,the variables that make up the primary key must be public.

The java.lang.Long class is also a good candidate for aprimary key class for the AccountBM bean.

## Using a database with a BMP entity bean

In an entity bean with BMP, each ejbFind method and all of the life cyclemethods (ejbActivate, ejbCreate, ejbLoad, ejbPassivate, and ejbStore) mustinteract with the data source (or sources) used by the bean to maintain itspersistent data. To interact with a supported database, the BMP entitybean must contain the code to manage database connections and to manipulatethe data in the database.The code required to manage database connections varies across the EJB serverimplementations:

- The EJB server (CB) uses JDBC 1.0 to manage database connectionsdirectly. For more information on the EJB server (CB), see Managing connections in the EJB server (CB) environment.
- The EJB server (AE) uses a set of specialized beans to encapsulateinformation about databases and an IBM-specific interface to JDBC to allowentity bean interaction with a connection manager. For more informationon the EJB server (AE), see Managing database connections in the EJB server (AE) environment.

In general, there are three approaches to getting and releasing connectionsto databases:

- The bean can get a database connection in the setEntityContext method andrelease it in the unsetEntityContext method. This approach is theeasiest for the enterprise bean developer to implement. However,without a connection manager, this approach is not viable because under itbean instances hold onto database connections even when they are not in use(that is, when the bean instance is passivated). Even with a connectionmanager, this approach does not scale well.
- The bean can get a database connection in the ejbActivate and ejbCreatemethods, get and release a database connection in each ejbFind method, andrelease the database connection in the ejbPassivate and ejbRemovemethods. This approach is somewhat more difficult to implement, but itensures that only those bean instances that are activated have connections tothe database. If you are using the EJB server (CB), which does notallow BMP entity beans to use the connection manager, this approach isprobably the best one.
- The bean can get and release a database connection in each method thatrequires a connection: ejbActivate, ejbCreate, ejbFind, ejbLoad, andejbStore. This approach is more difficult to implement than the firstapproach, but is no more difficult than the second approach. If you areusing the EJB server (AE), which contains a connection manager, this approachis the most efficient in terms of connection use and also the mostscalable.

The example AccountBM bean, uses the second approach described in thepreceding text. The AccountBMBean class contains two methods for makinga connection to the DB2 database, checkConnection and makeConnection, and onemethod to drop connections: dropConnection. These methods must becoded differently based on which EJB server environment you use:

- The code required to make the AccountBM bean work with the connectionmanager in the EJB server (CB) is shown in Managing connections in the EJB server (CB) environment.
- The code required to make the AccountBM bean work with the connectionmanager in the EJB server (AE) is shown in Managing database connections in the EJB server (AE) environment.

The code required to manipulate data in a database is identical for bothEJB server environments. For more information, see Manipulating data in a database.

### Managing connections in the EJB server (CB) environment

In the EJB server (CB) environment, both JDBC 1.0 connectivity (usingthe java.sql.DriverManager interface) and JDBC 2.0connectivity (using the javax.sql.DataSource interface) aresupported, although full JDBC 2.0 support requires DB2 version7.1, FixPack 2.

Under JDBC 2.0, database connections are made as described in Managing database connections in the EJB server (AE) environment. You must replace the Advanced Edition-specificcom.ibm.db2.jdbc.app.stdext.javax.sql.DataSourceinterface with the standard JDBC 2.0 interfacejavax.sql.DataSource interface. (When you are using DB27.1, FixPack 2, this is implemented by theCOM.ibm.db2.jdbc.DB2DataSource class, which anadministrator must bind into the JNDI namespace.)

Under JDBC 1.0, the java.sql.DriverManager interfaceis used to load and register a database driver and to get and releaseconnections to the database. This process is described in the rest ofthis section.

#### Loading and registering a data source

The example AccountBM bean uses an IBM DB2 relational database to store itspersistent data. To interact with DB2, the example bean must load oneof the available JDBC drivers. Figure 62 shows the code required to load the driver class. Thevalue of the *driverName* variable is obtained by the getEnvPropsmethod, which accesses a corresponding environment variable in the deployedenterprise bean.

The Class.forName method loads and registers the driverclass. The AccountBM bean loads the driver in its setEntityContextmethod, ensuring that every instance of the bean has immediate access to thedriver after creating the bean instance and establishing the bean'scontext.

**Note:**

In the EJB server (CB) environment, entity beans with BMP that use JDBC toaccess a database cannot participate in distributed transactions because theenvironment does not support XA-enabled JDBC.

**Figure 62. Code example: Loading and registering a JDBC driver in the setEntityContext method**

```
public void setEntityContext(EntityContext ctx)     throws EJBException {     entityContext = ctx;
try {          getEnvProps();          // Load the applet driver for DB2
Class.forName(driverName);     } catch (Exception e) {          ...     }}
```

#### Creating and closing a connection to a database

After loading and registering a database driver, the BMP entity bean must geta connection to the database. When it no longer needs that connection,the BMP entity bean must close the connection.

In the AccountBMBean class, the checkConnection method is called withinother bean class methods that require a database connection, but for which itcan be assumed that a connection already exists. This method checks tomake sure that the connection is still available by checking if the*jdbcConn* variable is set to null. If the variable is null,the makeConnection method is invoked to get the connection.

The makeConnection method is invoked when a new database connection isrequired. It invokes the static methodjava.sql.DriverManager.getConnection and passes the DB2URL value defined in the *jdbcUrl* variable (and described in Defining instance variables). The getConnection method is overloaded; themethod shown here only uses the database URL, other versions require the URLand the database user ID or the URL, database user ID, and the userpassword.

**Figure 63. Code example: The checkConnection and makeConnection methods of the AccountBMBean class**

```
# import java.sql.*;...private void checkConnection() throws EJBException {     if (jdbcConn ==
null) {          makeConnection();     }     return;}...private void makeConnection() throws
EJBException {     ...     try {          // Open database connection          jdbcConn =
DriverManager.getConnection(jdbcUrl);     } catch(Exception e) { // Could not get database
connection          ...     }}
```

Entity beans with BMP must also drop database connections when a particularbean instance no longer requires it. The AccountBMBean class contains adropConnection method to handle this task. To drop the databaseconnection, the dropConnection method does the following:

1. Invokes the commit method on the connection object (*jdbcConn*),to drop any locks held on the database.
2. Invokes the close method on the connection object to close theconnection.
3. Sets the connection object reference to null.

**Figure 64. Code example: The dropConnection method of the AccountBMBean class**

```
private void dropConnection() {     try {          // Close and delete jdbcConn
jdbcConn.commit();     } catch (Exception e) {          // Could not commit transactions to
database          ...     } finally {          jdbcConn.close();          jdbcConn = null;     }}
```

## Managing database connections in the EJB server (AE) environment

In the EJB server (AE) environment, the administrator creates a specializedset of entity beans that encapsulate information about the database and thedatabase driver. These specialized entity beans are created by usingthe WebSphere Administrative Console.

An entity bean that requires access to a database must use JNDI to create areference to an EJB object associated with the right database beaninstance. The entity bean can then use the IBM-specific interface(namedcom.ibm.db2.jdbc.app.stdext.javax.sql.DataSource)to get and release connections to the database.

The DataSource interface enables the entity bean to transparently interactwith the connection manager of the EJB server (AE). The connectionmanager creates a pool of database connections, which are allocated anddeallocated to individual entity beans as needed.

**Note:**

The example code contained in this section cannot be found in theAccountBMBean, which manages database connections by using the DriverManagerinterface described in Managing connections in the EJB server (CB) environment. This section shows the code that is required if theAccountBM bean were rewritten to use the DataSource interface.

#### Getting an EJB object reference to a data source bean instance

Before a BMP entity bean can get a connection to a database, the entity beanmust perform a JNDI lookup on the data source entity bean associated with thedatabase used to store the BMP entity bean's persistent data. Figure 65 shows the code required to create an InitialContext objectand then get an EJB object reference to a database bean instance. TheJNDI name of the database bean is defined by the administrator; it isrecommended that the JNDI naming convention be followed when defining thisname. The value of the required database-specific variables areobtained by the getEnvProps method, which accesses the correspondingenvironment variables from the deployed enterprise bean.

Because the connection manager creates and drops the actual databaseconnections and simply allocates and deallocates these connections asrequired, there is no need for the BMP entity bean to load and register thedatabase driver. Therefore, there is no need to define the*driverName* and *jdbcUrl* variables discussed in Defining instance variables.

**Figure 65. Code example: Getting an EJB object reference to a data source bean instance in the setEntityContext method (rewritten to use DataSource)**

```
...# import com.ibm.db2.jdbc.app.stdext.javax.sql.DataSource;# import
javax.naming.*;...InitialContext initContext = null;DataSource ds = null;...    public void
setEntityContext(EntityContext ctx)     throws EJBException {     entityContext = ctx;          try
{              getEnvProps();              ds = initContext.lookup("jdbc/sample");          }
catch (NamingException e) {              ...          }     }...
```

#### Allocating and deallocating a connection to a database

After creating an EJB object reference for the appropriate database beaninstance, that object reference is used to get and release connections to thecorresponding database. Unlike when using the DriverManager interface,when using the DataSource interface, the BMP entity bean does not reallycreate and close data connections; instead, the connection managerallocates and deallocates connections as required by the entity bean.Nevertheless, a BMP entity bean must still contain code to send allocation anddeallocation requests to the connection manager.

In the AccountBMBean class, the checkConnection method is called withinother bean class methods that require a database connection, but for which itcan be assumed that a connection already exists. This method checks tomake sure that the connection is still available by checking if the*jdbcConn* variable is set to null. If the variable is null,the makeConnection method is invoked to get the connection (that is aconnection allocation request is sent to the connection manager).

The makeConnection method is invoked when a database connection isrequired. It invokes the getConnection method on the data sourceobject. The getConnection method is overloaded: it can takeeither a user ID and password or no arguments, in which case the user ID andpassword are implicitly set to null (this version is used in Figure 66).

**Figure 66. Code example: The checkConnection and makeConnection methods of the AccountBMBean class (rewritten to use DataSource)**

```
private void checkConnection() throws EJBeException {     if (jdbcConn == null) {
makeConnection();     }     return;}...private void makeConnection() throws EJBeException {     ...
try {          // Open database connection          jdbcConn = ds.getConnection();     }
catch(Exception e) { // Could not get database connection          ...     }}
```

Entity beans with BMP must also release database connections when a particularbean instance no longer requires it (that is, they must send a deallocationrequest to the connection manager). The AccountBMBean class contains adropConnection method to handle this task. To release the databaseconnection, the dropConnection method does the following (as shown in Figure 67):

1. Invokes the close method on the connection object to tell the connectionmanager that the connection is no longer needed.
2. Sets the connection object reference to null.

Putting the close method inside a try/catch/finally block ensures that theconnection object reference is always set to null even if the close methodfails for some reason. Nothing is done in the catch block because theconnection manager must clean up idle connections; this is not the job ofthe enterprise bean code.

**Figure 67. Code example: The dropConnection method of the AccountBMBean class (rewritten to use DataSource)**

```
private void dropConnection() {     try {          // Close the connection
jdbcConn.close();     catch (SQLException ex) {          // Do nothing      } finally {
jdbcConn = null;     }}
```

## Manipulating data in a database

After an instance of a BMP entity bean obtains a connection to its database,it can read and write data. The AccountBMBean class communicates withthe DB2 database by constructing and executing Java Structured Query Language(JSQL) calls by using the java.sql.PreparedStatementinterface.

As shown in Figure 68, the SQL call is created as a String(*sqlString*). The String variable is passed to thejava.sql.Connection.prepareStatement method; and thevalues of each variable in the SQL call are set by using the various settermethods of the PreparedStatement class. (The variables are substitutedfor the question marks in the *sqlString* variable.) Invokingthe PreparedStatement.executeUpdate method executes the SQLcall.

**Figure 68. Code example: Constructing and executing an SQL update call in an ejbCreate method**

```
private void ejbCreate(AccountBMKey key, int type, float initialBalance)      throws
CreateException, EJBException {     // Initialize persistent variables and check for good DB
connection     ...     // INSERT into database     try {          String sqlString = "INSERT INTO "
+ tableName +              " (balance, type, accountid) VALUES (?,?,?)";          PreparedStatement
sqlStatement = jdbcConn.prepareStatement(sqlString);          sqlStatement.setFloat(1, balance);
sqlStatement.setInt(2, type);          sqlStatement.setLong(3, accountId);          // Execute query
int updateResults = sqlStatement.executeUpdate();          ...     }     catch (Exception e) { //
Error occurred during insert          ...     }     ...}
```

The executeUpdate method is called to insert or update data in adatabase; the executeQuery method is called to get data from adatabase. When data is retrieved from a database, the executeQuerymethod returns a java.sql.ResultSet object, which must beexamined and manipulated using the methods of that class.

**Note:**

To improve scalability and performance, you do not need to callPreparedStatement for each database update. Instead, you can cache theresults of the first PreparedStatement call.

Figure 69 provides an example of how the data in a ResultSet ismanipulated in the ejbLoad method of the AccountBMBean class.

**Figure 69. Code example: Manipulating a ResultSet object in the ejbLoad method**

```
public void ejbLoad () throws EJBeException {     // Get data from database     try {          //
SELECT from database          ...          // Execute query          ResultSet sqlResults =
sqlStatement.executeQuery();          // Advance cursor (there should be only one item)
sqlResults.next();          // Pull out results          balance = sqlResults.getFloat(1);
type = sqlResults.getInt(2);     } catch (Exception e) {          // Something happened while
loading data.          ...     }}
```

## Using bean-managed transactions

In most situations, an enterprise bean can depend on the container to managetransactions within the bean. In these situations, all you need to dois set the appropriate transactional properties in the deployment descriptoras described in Enabling transactions and security in enterprise beans.

Under certain circumstances, however, it can be necessary to have anenterprise bean participate directly in transactions. By setting the*transaction* attribute in an enterprise bean's deploymentdescriptor to TX_BEAN_MANAGED, you indicate to the container that the bean isan active participant in transactions.

**Note:**

The value TX_BEAN_MANAGED is not a valid value for the *transaction*deployment descriptor attribute in entity beans. In other words, entitybeans cannot manage transactions.

When writing the code required by an enterprise bean to manage its owntransactions, remember the following basic rules:

- An instance of a stateless session bean *cannot* reuse the sametransaction context across multiple methods called by an EJB client.Therefore, it is recommended that the transaction context be a local variableto each method that requires a transaction context.
- An instance of a stateful session bean can reuse the same transactioncontext across multiple methods called by an EJB client. Therefore,make the transaction context an instance variable or a local method variableat your discretion. (When a transaction spans multiple methods, you canuse the javax.ejb.SessionSynchronization interface tosynchronize the conversational state with the transaction.)

**Note:**

In the EJB server (CB) environment, a stateful session bean that implementsthe TX_BEAN_MANAGED attribute must begin and complete a transaction within thescope of a single method.

Figure 70 shows the standard code required to obtain an objectencapsulating the transaction context. There are three basics stepsinvolved:

1. The enterprise bean class must set the value of thejavax.ejb.SessionContext object reference in thesetSessionContext method.
2. A javax.transaction.UserTransaction object is created bycalling the getUserTransaction method on the SessionContext objectreference.
3. The UserTransaction object is used to participate in the transaction bycalling transaction methods such as begin and commit as needed. If aenterprise bean begins a transaction, it must also complete that transactioneither by invoking the commit method or the rollback method.

   **Note:**

   In both EJB servers, the getUserTransaction method of thejavax.ejb.EJBContext interface (which is inherited by theSessionContext interface) returns an object of typejavax.transaction.UserTransaction rather than typejavax.jts.UserTransaction. While this is a deviation fromthe 1.0 version of the EJB Specification, the 1.1 version of theEJB Specification requires that the getUserTransaction method return an objectof type javax.transaction.UserTransaction and drops therequirement to return objects of typejavax.jts.UserTransaction.

**Figure 70. Code example: Getting an object that encapsulates a transaction context**

```
...import javax.transaction.*;...public class MyStatelessSessionBean implements SessionBean {
private SessionContext mySessionCtx = null;     ...     public void
setSessionContext(.SessionContext ctx) throws EJBException {          mySessionCtx = ctx;      }
...     public float doSomething(long arg1) throws FinderException, EJBException {
UserTransaction userTran = mySessionCtx.getUserTransaction();          ...          // User userTran
object to call transaction methods          userTran.begin();          // Do transactional work
...          userTran.commit();          ...     }     ...}
```

The following methods are available with the UserTransactioninterface:

- begin--Begins a transaction. This method takes no argumentsand returns void.
- commit--Attempts to commit a transaction; assuming that nothingcauses the transaction to be rolled back, successful completion of this methodcommits the transaction. This method takes no arguments and returnsvoid.
- getStatus--Returns the status of the referenced transaction.This method takes no arguments and returns int; if no transaction isassociated with the reference, STATUS_NO_TRANSACTION is returned. Thefollowing are the valid return values for this method:
  - STATUS_ACTIVE--Indicates that transaction processing is still inprogress.

- STATUS_COMMITTED--Indicates that a transaction has been committed andthe effects of the transaction have been made permanent.
- STATUS_COMMITTING--Indicates that a transaction is in the process ofcommitting (that is, the transaction has started committing but has notcompleted the process).
- STATUS_MARKED_ROLLBACK--Indicates that a transaction is marked to berolled back.
- STATUS_NO_TRANSACTION--Indicates that a transaction does not exist inthe current transaction context.
- STATUS_PREPARED--Indicates that a transaction has been prepared butnot completed.
- STATUS_PREPARING--Indicates that a transaction is in the process ofpreparing (that is, the transaction has started preparing but has notcompleted the process).
- STATUS_ROLLEDBACK--Indicates that a transaction has been rolledback.
- STATUS_ROLLING_BACK--Indicates that a transaction is in the processof rolling back (that is, the transaction has started rolling back but has notcompleted the process).
- STATUS_UNKNOWN--Indicates that the status of a transaction isunknown.

- rollback--Rolls back the referenced transaction. This methodtakes no arguments and returns void.
- setRollbackOnly--Specifies that the only possible outcome of thetransaction is rollback. This method takes no arguments and returnsvoid.
- setTransactionTimeout--Sets the timeout (in seconds) associated withthe transaction. If some transaction participant has not specificallyset this value, a default timeout is used. This method takes a numberof seconds (as type int) and returns void.

# Enabling transactions and security in enterprise beans

This chapter examines how to enable transactions and security in enterprisebeans by setting the appropriate deployment descriptor attributes:

- For transactions, a session bean can either use container-managedtransactions or implement bean-managed transactions; entity beans mustuse container-managed transactions. To enable container-managedtransactions, you must set the transaction attribute to any value*except* TX_BEAN_MANAGED and set the transaction isolation levelattribute. To enable bean-managed transactions, you must set thetransaction attribute to TX_BEAN_MANAGED and set the transaction isolationlevel attribute. For more information, see Setting transactional attributes in the deployment descriptor.

  If you want a session bean to manage its own transactions, you must writethe code that explicitly demarcates the boundaries of a transaction asdescribed in Using bean-managed transactions.

  If you want an EJB client to manage its own transactions, you mustexplicitly code that client to do so as described in Managing transactions in an EJB client.

- For security, the *run-as mode* attribute is used by the EJBserver environments. For information on the valid values of thisattribute, see Setting the security attribute in the deployment descriptor.

These attributes, like the other deployment descriptor attributes, are setby using one of the tools available with either the EJB server (AE) or the EJBserver (CB). For more information, see Tools for developing and deploying enterprise beans in the EJB server (AE) environment or Tools for developing and deploying enterprise beans in the EJB server (CB) environment.

---

## Setting transactional attributes in the deployment descriptor

The EJB Specification describes the creation of applications that enforcetransactional consistency on the data manipulated by the enterprisebeans. However, unlike other specifications that support distributedtransactions, the EJB specification does not require enterprise bean and EJBclient developers to write any special code to use transactions.Instead, the container manages transactions based on two deployment descriptorattributes associated with the EJB module, and the enterprise bean and EJBapplication developers are freed to deal with the business logic of theirapplications.

Enterprise bean developers can specifically design enterprise beans and EJBapplications that explicitly manage transactions. For more information,see Using bean-managed transactions.

Under most conditions, transaction management can be handled within theenterprise beans, freeing the EJB client developer of this task.However, EJB clients can participate in transactions if required ordesired. For more information, see Managing transactions in an EJB client.

Two attributes determine the way in which an enterprise bean is managedfrom a transactional perspective:

- The *transaction* attribute defines the transactional manner inwhich the container invokes a method. This attribute is part of thestandard deployment descriptor. Setting the transaction attribute defines the valid values of this attribute and explainstheir meanings.
- The *transaction isolation level* attribute defines the manner inwhich transactions are isolated from each other by the container. Thisattribute is an extension to the standard deployment descriptor. Setting the transaction isolation level attribute defines the valid values of this attribute and explainstheir meanings.

## Setting the transaction attribute

The transaction attribute defines the transactional manner in which thecontainer invokes enterprise bean methods. This attribute is set forindividual methods in a bean.

**Note:**

The EJB server (CB) does not support the setting of the transaction attributefor individual enterprise bean methods; the transaction attribute can beset only for the entire bean.

The following are valid values for this attribute in decreasing order oftransactional strictness:

**TX_BEAN_MANAGED**

Notifies the container that the bean class directly handles transactiondemarcation. This attribute value can be specified only for sessionbeans and it cannot be specified for individual bean methods. For moreinformation on designing session beans to implement this attribute value, see Using bean-managed transactions.

In the EJB server (CB) environment, if a stateful session bean has thisattribute value, a method that begins a transaction must also complete thattransaction (commit or roll back the transaction). In other words, atransaction cannot span multiple methods in a stateful session bean when usedin the EJB server (CB) environment.

**TX_MANDATORY**

Directs the container to always invoke the bean method within thetransaction context associated with the client. If the client attemptsto invoke the bean method without a transaction context, the container throwsthe javax.jts.TransactionRequiredException exception to theclient. The transaction context is passed to any EJB object or resourceaccessed by an enterprise bean method.

EJB clients that access these entity beans must do so within an existingtransaction. For other enterprise beans, the enterprise bean or beanmethod must implement the TX_BEAN_MANAGED value or use the TX_REQUIRED orTX_REQUIRES_NEW value. For non-enterprise bean EJB clients, the clientmust invoke a transaction by using thejavax.transaction.UserTransaction interface, as described in Managing transactions in an EJB client.

In the EJB server (CB) environment, this attribute value must be used inentity beans with container-managed persistence (CMP) that use Host On-Demand(HOD) or the External Call Interface (ECI) to access CICS or IMSapplications.

**TX_REQUIRED**

Directs the container to invoke the bean method within a transactioncontext. If a client invokes a bean method from within a transactioncontext, the container invokes the bean method within the client transactioncontext. If a client invokes a bean method outside of a transactioncontext, the container creates a new transaction context and invokes the beanmethod from within that context. The transaction context is passed toany enterprise bean objects or resources that are used by this beanmethod.

**TX_REQUIRES_NEW**

Directs the container to always invoke the bean method within a newtransaction context, regardless of whether the client invokes the methodwithin or outside of a transaction context. The transaction context ispassed to any enterprise bean objects or resources that are used by this beanmethod.

The EJB server (CB) does *not* support this attribute value forenterprise beans written to version 1.0 of the EJBspecification. It interprets the TX_REQUIRES_NEW attribute asTX_REQUIRED for Enterprise beans written to version 1.1 of the EJBspecification.

**TX_SUPPORTS**

Directs the container to invoke the bean method within a transactioncontext if the client invokes the

bean method within a transaction. Ifthe client invokes the bean method without a transaction context, thecontainer invokes the bean method without a transaction context. Thetransaction context is passed to any enterprise bean objects or resources thatare used by this bean method.

In the EJB server (CB) environment, entity beans with CMP must be accessedwithin a transaction. If an entity bean with CMP uses this transactionattribute, the EJB client must initiate a transaction before invoking a methodon the entity bean.

**TX_NOT_SUPPORTED**

Directs the container to invoke bean methods without a transactioncontext. If a client invokes a bean method from within a transactioncontext, the container suspends the association between the transaction andthe current thread before invoking the method on the enterprise beaninstance. The container then resumes the suspended association when themethod invocation returns. The suspended transaction context is*not* passed to any enterprise bean objects or resources that areused by this bean method.

In the EJB server (CB) environment, entity beans with CMP must be accessedwithin a transaction. Therefore, this attribute value is not supportedin entity beans with CMP in the EJB server (CB) environment.

**TX_NEVER**

Directs the container to invoke bean methods without a transactioncontext.

- If the client invokes a bean method from within a transaction context, thecontainer throws the java.rmi.RemoteException exception.
- If the client invokes a bean method from outside a transaction context,the container behaves in the same way as if the TX_NOT_SUPPORTED transactionattribute was set. The client must call the method without atransaction context.

In the EJB server (CB) environment, the TX_NEVER attribute is interpretedas TX_NOT_SUPPORTED. Therefore, no exception is thrown if the clientinvokes a bean method from within a transaction context.

**Table 3. Effect of the enterprise bean's transaction attribute on the transaction context**

| Transaction attribute | Client transaction context | Bean transaction context |
|---|---|---|
| TX_MANDATORY | No transaction | Not allowed |
| | Client transaction | Client transaction |
| TX_REQUIRES_NEW | No transaction | New transaction |
| | Client transaction | New transaction |
| TX_REQUIRED | No transaction | New transaction |
| | Client transaction | Client transaction |
| TX_SUPPORTS | No transaction | No transaction |
| | Client transaction | Client transaction |
| TX_NOT_SUPPORTED | No transaction | No transaction |
| | Client transaction | No transaction |
| TX_NEVER | No transaction | No transaction |
| | No transaction | No transaction |

When setting the deployment descriptor for an entity bean, you can markgetter methods as "Read-Only" methods to improve performance. If atransaction unit of work includes no methods other than "Read-Only" designatedmethods, then the entity bean state synchronization does not invokestore.

## Setting the transaction isolation level attribute

**Note:**

The EJB server (CB) does not support the transaction isolation levelattribute.

The transaction isolation level determines how strongly one transaction isisolated from another. This attribute is set for individual methods ina bean. However, within a transactional context, the isolation levelassociated with the first method invocation becomes the required isolationlevel for all other methods invoked within that transaction. If amethod is invoked with a different isolation level from that of the firstmethod, the java.rmi.RemoteException exception is thrown.

The following are valid values for this attribute, in decreasing order ofisolation:

**TRANSACTION_SERIALIZABLE**

This level prohibits all of the following types of reads:

- *Dirty reads*, where a transaction reads a database rowcontaining uncommitted changes from a second transaction.
- *Nonrepeatable reads*, where one transaction reads a row, asecond transaction changes the same row, and the first transaction rereads therow and gets a different value.
- *Phantom reads*, where one transaction reads all rows thatsatisfy an SQL WHERE condition, a second transaction inserts a row that alsosatisfies the WHERE condition, and the first transaction applies the sameWHERE condition and gets the row inserted by the second transaction.

**TRANSACTION_REPEATABLE_READ**

This level prohibits dirty reads and nonrepeatable reads, but it allowsphantom reads.

**TRANSACTION_READ_COMMITTED**

This level prohibits dirty reads, but allows nonrepeatable reads andphantom reads.

**TRANSACTION_READ_UNCOMMITTED**

This level allows dirty reads, nonrepeatable reads, and phantomreads.

These isolation levels correspond to the isolation levels defined in theJava Database Connectivity (JDBC) java.sql.Connectioninterface.

The container uses the transaction isolation level attribute asfollows:

- Session beans and entity beans with bean-managed persistence(BMP)--For each database connection used by the bean, the container setsthe transaction isolation level at the start of each transaction.
- Entity beans with container-managed persistence (CMP)--The containergenerates database access code that implements the specified isolationlevel.

None of these values permits two transactions to update the same dataconcurrently; one transaction must end before another can update the samedata. These values determine only how locks are managed for readingdata. However, risks to consistency can arise from read operations whena transaction does further work based on the values read. For example,if one transaction is updating a piece of data and a second transaction ispermitted to read that data after it has been changed but before the updatingtransaction ends, the reading transaction can make a decision based on achange that is eventually rolled back. The second transaction risksmaking a decision on transient data.

Deciding which isolation level to use depends on several factors:

- The acceptable level of risk to data consistency
- The acceptable levels of concurrency and performance
- The isolation levels supported by the underlying database

The first two factors, risk to consistency and level of concurrency, arerelated. Decreasing the risk to consistency requires you to decreaseconcurrency because reducing the risk to consistency requires holding lockslonger. The longer a lock is held on a piece of data, the longerconcurrently running transactions must wait to access that data. TheTRANSACTION_SERIALIZABLE value protects data by eliminating concurrent accessto it. Conversely, the TRANSACTION_READ_UNCOMMITTED value allows thehighest degree of concurrency but entails the greatest risk toconsistency. You need to balance these two factors appropriately foryour application.

By default, most developers deploy enterprise beans with the transactionisolation level set to TRANSACTION_SERIALIZABLE. This is the defaultvalue in IBM VisualAge for Java Enterprise Edition and other deploymenttools. It is also the most restrictive and protected transactionisolation level incurring the most overhead. Some workloads do notrequire the isolation level and protection afforded byTRANSACTION_SERIALIZABLE. A given application might never update theunderlying data or be run with other applications that also make concurrentupdates. In that case, the application would not have to be concernedwith dirty, non-repeatable, or phantom reads. TheTRANSACTION_READ_UNCOMMITTED isolation level would probably besufficient.

Because the transaction isolation level is set in the EJB module'sdeployment descriptor, the same enterprise bean could be reused in differentapplications with different transaction isolation levels. The isolationlevel requirements should be reviewed and adjusted appropriately to increaseperformance.

The third factor, isolation levels supported in the database, means thatalthough the EJB specification allows you to request one of the four levels oftransaction isolation, it is possible that the database being used in theapplication does not support all of the levels. Also, vendors ofdatabase products implement isolation levels differently, so the precisebehavior of an application can vary from database to database. You needto consider the database and the isolation levels it supports when deciding onthe value for the transaction isolation attribute in deploymentdescriptors. Consult your database documentation for more informationon supported isolation levels.

---

## Setting the security attribute in the deployment descriptor

When an EJB client invokes a method on an enterprise bean, the user context ofthe client principal is encapsulated in a CORBA Current object, which containscredential properties for the principal. The Current object is passedamong the participants in the method invocation as required to complete themethod.

The security service uses the credential information to determine thepermissions that a principal has on various resources. At appropriatepoints, the security service determines if the principal is authorized to usea particular resource based on the principal's permissions.

If the method invocation is authorized, the security service does thefollowing with the principal's credential properties based on the valueof the *run-as mode* attribute of the enterprise bean. If aspecific identity is required, the *RunAsIdentity* attribute is usedto specify that identity.

**CLIENT_IDENTITY**

The security service makes no changes to the principal's credentialproperties.

**SYSTEM_IDENTITY**

The security service alters the principal's credential properties tomatch the credential properties associated with the EJB server.

**SPECIFIED_IDENTITY**

The security service attempts to match the principal's credentialproperties with the identity of any application with which the enterprise beanis associated. If successful, the security service alters theprincipal's credential properties to match the credential properties ofthe application.

# Developing servlets that use enterprise beans

A servlet is a Java application that enables users to access Web serverfunctionality. To use servlets, a Web server is required. TheWebSphere Application Server plugs into a number of commonly used Webservers. In addition, the IBM HTTP Web server is available with boththe Advanced Application Server and the Enterprise Application Server.For more information, consult the Advanced Edition InfoCenter.

Java servlets can be combined with enterprise beans to create powerful EJBapplications. This chapter describes how to use enterprise beans withina servlet. The example CreateAccount servlet, which uses the exampleAccount bean, is used to illustrate the concepts discussed in thischapter. The example servlet and enterprise bean discussed in thischapter are explained in Information about the examples described in the documentation.

---

## An overview of standard servlet methods

Usually, a servlet is invoked from an HTML form on the user'sbrowser. The first time the servlet is invoked, the servlet's initmethod is run to perform any initializations required at startup. Forthe first and all subsequent invocations of the servlet, the doGet method (or,alternatively, the doPost method) is run. Within the doGet method (orthe doPost method), the servlet gets the information provided by the user onthe HTML form and uses that information to perform work on the server andaccess server resources.

The servlet then prepares a response and sends the response back to theuser. After a servlet is loaded, it can handle multiple simultaneoususer requests. Multiple request threads can invoke the doGet (ordoPost) method at the same time, so the servlet needs to be made threadsafe.

When a servlet shuts down, the destroy method of the servlet is run inorder to perform any needed shutdown processing.

---

## Writing an HTML page that embeds a servlet

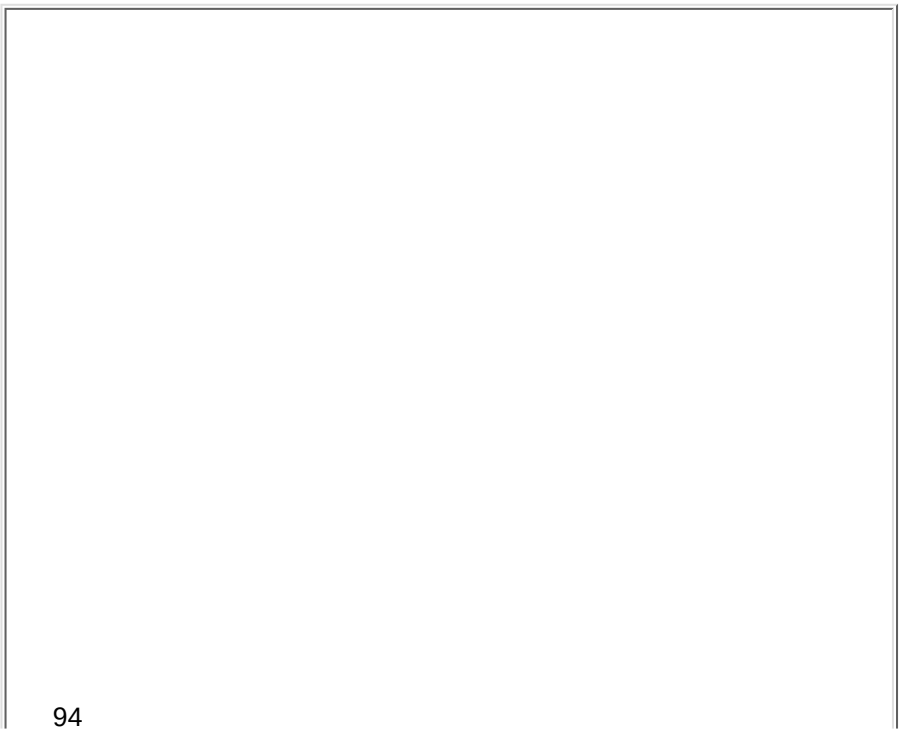
Figure 46 shows the HTML file (named create.html) used toinvoke the CreateAccount servlet. The HTML form is used to specify theaccount number for the new account, its type (checking or savings), and itsinitial balance. The request is passed to the doGet method of theservlet, where the servlet is identified with its full Java package name, asshown in the example.

**Figure 46. Code example: Content of the create.html file used to access the CreateAccount servlet**

```
<html><head><title>Create a new Account</title></head><body><h1 align="center">Create a new
Account</h1><form method="get"action="/servlet/com.ibm.ejs.doc.client.CreateAccount"><table border
align="center"><!-- specify a new account number --><tr bgcolor="#cccccc"><td align="right">Account
Number:</td><td colspan="2"><input type="text" name="account" size="20"maxlength="10"></tr><!--
specify savings or checking account -->...<!-- specify account starting balance -->...<!-- submit
information to servlet -->...<input type="submit" name ="submit" value="Create">...<!-- message area
-->...</form></body></html>
```

The HTML response from the servlet is designed to produce a displayidentical to create.html, enabling the user to continue creating newaccounts. Figure 47 shows what create.html looks like on abrowser.

**Figure 47. The initial form and output of the CreateAccount servlet**




## Developing the servlet

This section discusses the basic code required by a servlet that interactswith an enterprise bean. Figure 48 shows the basic outline of the code that makes up theCreateAccount servlet. As shown in the example, the CreateAccountservlet extends the javax.servlet.http.HttpServlet classand implements an init method and a doGet method.

**Figure 48. Code example: The CreateAccount class**

```
package com.ibm.ejs.doc.client;// General enterprise bean code.import
java.rmi.RemoteException;import javax.ejb.DuplicateKeyException;// Enterprise bean code specific to
this servlet.import com.ibm.ejs.doc.account.AccountHome;import
com.ibm.ejs.doc.account.AccountKey;import com.ibm.ejs.doc.account.Account;// Servlet related.import
javax.servlet.*;import javax.servlet.http.*;// JNDI (naming).import javax.naming.*;  // for Context,
InitialContext, NamingException// Miscellaneous:import java.util.*;import java.io.*;...public class
CreateAccount extends HttpServlet {   // Variables    ...   public void init(ServletConfig config)
throws ServletException {      ...   }   public void doGet(HttpServletRequest req,
HttpServletResponse res)   throws ServletException, IOException {      // --- Read and validate user
input, initialize. ---      ...      // --- If input parameters are good, try to create account. ---
...      // --- Prepare message to accompany response. ---      ...      // --- Prepare and send
HTML response. ---      ...}
```

### The servlet's instance variables

Figure 49 shows the instance variables used in the CreateAccountservlet. The *nameService*, *accountName*, and*providerUrl* variables are used to specify the property valuesrequired during JNDI lookup. These values are obtained from theClientResourceBundle class as described in Creating and getting a reference to a bean's EJB object.

The CreateAccount class also initializes the string constants that are usedto create the HTML response sent back to the user. (Only three of thesevariables are shown, but there are many of them). The init method inthe CreateAccount servlet provides a way to read strings from a resourcebundle to override these US English defaults in order to provide a response ina different national language.The instance variable *accountHome* is used by all client requests tocreate a new Account bean instance. The *accountHome* variableis initialized in the init method as shown in Figure 49.

**Figure 49. Code example: The instance variables of the CreateAccount class**

```
...public class CreateAccount extends HttpServlet {     // Variables for finding the home
private String nameService = null;     private String accountName = null;      private String
providerURL = null;     private ResourceBundle bundle =  ResourceBundle.getBundle(
"com.ibm.ejs.doc.client.ClientResourceBundle");     // Strings for HTML output - US English defaults
shown.     static String title = "Create a new Account";     static String number  = "Account
Number:";     static String type  = "Type:";     ...     // Variable for accessing the enterprise
bean.     private AccountHome accountHome = null;     ...     }
```

## The servlet's init method

The init method of the CreateAccount servlet is shown in Figure 50. The init method is run once, the first time arequest is processed by the servlet, after the servlet is started.Typically, the init method is used to do any one-time initializations for aservlet. For example, the default US English strings used in preparingthe HTML response can be replaced with another national language.The init method is also the best place to initialize the value of referencesto the home interface of any enterprise beans used by the servlet. Inthe CreateAccount's init method, the *accountHome* variable isinitialized to reference the EJB home object of the Account bean.

As in other types of EJB clients, the properties required to do a JNDIlookup are specific to the EJB implementation. Therefore, theseproperties are externalized in a properties file or a resource bundleclass. For more information on these properties, see Creating and getting a reference to a bean's EJB object.

Note that in the CreateAccount servlet, a HashTable object is used to storethe properties required to do a JNDI lookup whereas a Properties object isused in the TransferApplication. Both of these classes are valid forstoring these properties.

**Figure 50. Code example: The init method of the CreateAccount servlet**

```
// Variables for finding the EJB home objectprivate String nameService = null;private String
accountName = null;private String providerURL = null;private ResourceBundle bundle =
ResourceBundle.getBundle(          "com.ibm.ejs.doc.client.TransferResourceBundle");...public void
init(ServletConfig config) throws ServletException {     super.init(config);     ...     try {
// Get NLS strings from an external resource bundle          ...          createTitle =
bundle.getString("createTitle");          number = bundle.getString("number");          type =
bundle.getString("type");          ...          //Get values for the naming factory and home name.
nameService = bundle.getString("nameService");          accountName =
bundle.getString("accountName");          providerURL = bundle.getString("providerURL");     }
catch (Exception e) {          ...     }     // Get home object for access to Account enterprise
bean.     Hashtable env = new Hashtable();     env.put(Context.INITIAL_CONTEXT_FACTORY,
nameService);     try {          // Create the initial context.          Context ctx = new
InitialContext(env);          // Get the home object.          Object homeObject =
ctx.lookup(accountName);          // Get the AccountHome object.          accountHome =
(AccountHome) javax.rmi.PortableRemoteObject.narrow(
(org.omg.CORBA.Object)homeObject, AccountHome.class);     }     // Determine cause of failure.
catch (NamingException e) {          ...      }      catch (Exception e) {          ...     }}
```

## The servlet's doGet method

The doGet method is invoked for every servlet request. In theCreateAccount servlet, the method does the following tasks to manage userinput. These tasks are fairly standard for this method:

- Read the user input from the HTML form and decide if the input isvalid--for example, whether the user entered a valid number for aninitial balance.
- Perform the initializations required for each request.

Figure 51 shows the parts of the doGet method that handle userinput. Note that the *req* variable is used to read the userinput from the HTML form. The *req* variable is ajavax.servlet.http.HttpServletRequest object passed asone of the arguments to the doGet method.

**Figure 51. Code example: The doGet method of the CreateAccount servlet**

```
public void doGet (HttpServletRequest req, HttpServletResponse res)      throws ServletException,
IOException {     // --- Read and validate user input, initialize. ---     // Error flags.
boolean accountFlag  = true;     boolean balanceFlag  = true;     boolean inputFlag = false;
boolean createFlag  = true;     boolean duplicateFlag = false;     // Datatypes used to create new
account bean.     AccountKey key;     int typeAcct  = 0;     String typeString = "0";     float
initialBalance = 0;     // Read input parameters from HTML form.     String[] accountArray =
req.getParameterValues("account");     String[] typeArray = req.getParameterValues("type");
String[] balanceArray = req.getParameterValues("balance");     // Convert input parameters to needed
datatypes for new account.     // (account)     long accountLong = 0;     ...      key = new
AccountKey(accountLong);     // (type)     if (typeArray[0].equals("1")) {          typeAcct   = 1;
// Savings account.          typeString = "savings";     }     else if (typeArray[0].equals("2")) {
typeAcct   = 2;          // Checking account          typeString = "checking";     }     //
(balance)     try {          initialBalance = (Float.valueOf(balanceArray[0])).floatValue();     }
catch (Exception e) {          balanceFlag = false;     }     ...      // --- If input parameters
are good, try to create account bean. ---     ...     // --- Prepare message to accompany response.
---     ...     // --- Prepare and send HTML response. ---     ...}
```

## Creating an enterprise bean

If the user input is valid, the doGet method attempts to create a new accountbased on the user input as shown in Figure 52. Besides the initialization of the home objectreference in the init method, this is the only other piece of code that isspecific to the use of enterprise beans in a servlet.

**Figure 52. Code example: Creating an enterprise bean in the doGet method**

```
public void doGet(HttpServletRequest req, HttpServletResponse res)      throws ServletException,
IOException {     // --- Read and validate user input, initialize ---.     ...     // --- If input
parameters are good, try to create account bean. ---     if (accountFlag && balanceFlag) {
inputFlag = true;          try {               // Create the bean.               Account account =
accountHome.create(key, typeAcct, initialBalance);          }          // Determine cause of
failure.          catch (RemoteException e) {               ...          }          catch
(DuplicateKeyException e) {               ...          }          catch (Exception e) {
...          }     }     // --- Prepare message to accompany response. ---     ...     // ---
Prepare and send HTML response. ---     ... }
```

## Determining the content of the user response

Next, the doGet method prepares a response message to be sent to theuser. There are three possible responses:

- The user input was not valid.
- The user input was valid, but the account was not created for somereason.
- The account was created successfully. If the previous two errors donot occur, this response is prepared.

Figure 53 shows the code used by the servlet to determine whichresponse to send to the user. If no errors are encountered, then theresponse indicates success.

**Figure 53. Code example: Determining a user response in the doGet method**

```
public void doGet(HttpServletRequest req, HttpServletResponse res)      throws ServletException,
IOException {       // --- Read and validate user input, initialize. ---      ...      // --- If
input parameters are good, try to create account bean. ---      ...      // --- Prepare message to
accompany response. ---      ...     String messageLine  = "";     if (inputFlag) {          // If you
are here, the client input is good.          if (createFlag) {          // New account enterprise
bean was created.          messageLine =  createdaccount + " " + accountArray[0] + ", " +
createdtype + " " + typeString + ", " +                    createdbalance + " " + balanceArray[0];
}          else if (duplicateFlag) {               // Account with same key already exists.
messageLine = failureexists + " " + accountArray[0];          }          else {          // Other
reason for failure.               messageLine = failureinternal + " " + accountArray[0];          }
}     else {          // If you are here, something was wrong with the client input.          String
separator = "";          if (!accountFlag) {               messageLine = failureaccount + " " +
accountArray[0];               separator = ", ";     }     if (!balanceFlag) {          messageLine
= messageLine + separator +                    failurebalance + " " + balanceArray[0];     }     //
--- Prepare and send HTML response. ---     ... }
```

## Sending the user response

With the type of response determined, the doGet method then prepares the fullHTML response and sends it to the user's browser, incorporating theappropriate message. Relevant parts of the full HTML response are shownin Figure 54.The *res* variable is used to pass the response back to theuser.

This variable is an HttpServletResponse object passed as anargument to the doGet method. The response code shown here mixes bothdisplay (HTML) and content in one servlet. You can separate the displayand the content by using JavaServer Pages (JSP). A JSP allows thedisplay and content to be developed and maintained separately.

**Figure 54. Code example: Responding to the user in the doGet method**

```
public void doGet(HttpServletRequest req, HttpServletResponse res)      throws ServletException,
IOException {      // --- Read and validate user input, initialize. ---      ...      // --- If
input parameters are good, try to create account bean. ---      ...      // --- Prepare message to
accompany response. ---      ...      // --- Prepare and send HTML response. ---      // HTML returned
looks like initial HTML that invoked this servlet.      // Message line says whether servlet was
successful or not.      res.setContentType("text/html");      res.setHeader("Pragma", "no-cache");
res.setHeader("Cache-control", "no-cache");      PrintWriter out = res.getWriter();
out.println("<html>");      ...      out.println("<title> " + createTitle + "</title>");      ...
out.println(" </html>");   }
```

## Threading issues

Except for the instance variable required to get a reference to the Accountbean's home interface and to support multiple languages (which remainunchanged for all user requests), all other variables used in theCreateAccount servlet are local to the doGet method. Each requestthread has its own set of local variables, so the servlet can handlesimultaneous user requests.

As a result, the CreateAccount servlet is thread safe. By taking asimilar approach to servlet design, you can also make your servlets threadsafe.

# Tools for developing and deploying enterprise beans in the EJB server (CB) environment

The following are the basic approaches to developing and deploying enterprisebeans in the EJB server (CB) environment:

- You can use the tools available in the Java Software Development Kit (SDK)and WebSphere Application Server, Enterprise Edition. For moreinformation, see Developing and deploying enterprise beans with EJB server (CB) tools.
- You can use one of the available integrated development environments(IDEs) such as IBM VisualAge for Java. IDE tools automatically generatesignificant parts of the enterprise bean code and contain integrated tools forpackaging and testing enterprise beans. For more information, see Using VisualAge for Java.
- You can create an enterprise bean from an existing CICS or InformationManagement System (IMS) application by using the **PAOToEJB**tool. The application must be mapped into a procedural adapter object(PAO) before this tool is used. For more information, see Creating an enterprise bean from an existing CICS or IMS application.
- You can create an enterprise bean that communicates with IBM MQSeries byusing the **mqaaejb** tool. For more information, see Creating an enterprise bean that communicates with MQSeries.

Before beginning development of enterprise beans in the EJB server (CB)environment, review the list of development restrictions contained in Restrictions in the EJB server (CB) environment.

**Note:**

Deployment and use of enterprise beans for the EJB server (CB) environmentmust take place on the Microsoft Windows NT or Windows 2000 operating system,the IBM AIX operating systems, or the Sun Solaris operating system.

For information on developing and deploying enterprise beans in the EJBserver (AE) environment, see Tools for developing and deploying enterprise beans in the EJB server (AE) environment.

---

## Developing and deploying enterprise beans with EJB server (CB) tools

You need the following tools to develop and deploy enterprise beans for theEJB server (CB):

- An ASCII text editor. (You can use also use a Java development toolthat does not support enterprise bean development.)
- The SDK Java compiler (**javac**) and Java Archiving tool(**jar**).
- The following tools available in the WebSphere Application Server,Enterprise Edition:
  - **jetace**, which enables you to create or update an EJB JAR filefor one or more enterprise beans; this includes the creation of theenterprise bean's deployment descriptor, which instructs the EJB serveron how to properly manage the enterprise bean.

    **jetace** can only be used to create JAR files that are compatiblewith version 1.0 of the EJB specification. If you need to workwith JAR files compatible with version 1.1, see Using the Application Assembly Tool.
  - Object Builder, which is the recommended tool for deploying enterprisebeans. Use of this tool is not documented in this book. For moreinformation on using Object Builder to deploy enterprise beans, see theComponent Broker Application Development ToolsGuide.
  - **cbejb**, which works with Object Builder to create and compilethe necessary files needed by the EJB server (CB) to manage an enterprisebean. The **cbejb** tool looks inside the EJB JAR file toexamine the EJB home and EJB object classes and the deploymentdescriptors. The **cbejb** tool generates a model that ObjectBuilder uses to create the necessary deployment library files. Theoutput of this process is a set of server-side and client-side JAR and libraryfiles.
  - **CBDeployJar**, which automates the deployment of enterprisebeans. The **CBDeployJar** tool can be used to deploy JAR filesthat are compatible with either version 1.0 or version 1.1 ofthe EJB specification. It runs the **cbejb** tool to deploy thefiles, generates database table mappings for enterprise beans with CMP,compiles the deployed files, and configures and starts the EJB server.It also registers references to enterprise beans that are compatible withversion 1.1 in the JNDI namespace.
  - **CBDeployEar**, which is used to deploy enterprise beans from aJAR file stored in a Java(TM) 2 Enterprise Edition (J2EE(TM)) EnterpriseArchive (EAR) file. The **CBDeployEar** tool extracts a JAR filefrom an EAR file, then runs the **CBDeployJar** tool on the extractedJAR file.
  - **ejbbind**, which binds an enterprise bean's Java Naming andDirectory Interface (JNDI) home name (found in its deployment descriptor) to afactory in an EJB server (CB). This tool is deprecated for serversrunning on the AIX, Windows NT, Windows 2000, and Solaris platforms.
  - **appbind**, which allows enterprise bean deployers to create anapplication-specific naming context and associate it with a selected factoryfinder so that the EJB home lookup operations are resolved with that factoryfinder. This tool is available only on the AIX, Windows NT, Windows2000, and Solaris platforms and can only be applied to servers installed onany of those platforms.

This section describes the steps you must follow to develop and deployenterprise beans by using the EJB server (CB) tools. The followingtasks are involved:

1. Ensure that you have the prerequisite software to develop and deployenterprise beans in the EJB server (CB). For more information, see Prerequisite software for the EJB server (CB).
2. Set the CLASSPATH environment variable required by different components ofthe EJB server (CB) environment. For more information, see Setting the CLASSPATH environment variable in the EJB server (CB) environment.
3. Write and compile the components of the enterprise bean. For moreinformation, see Creating the components of an enterprise bean.
4. Create a finder helper class for each entity bean with CMP that containsspecialized finder methods (other than the findByPrimaryKey method).For more information, see Creating finder logic in the EJB server (CB).

5. Use the **jetace** tool to create an EJB JAR file to contain theenterprise bean. For more information, see Creating an EJB JAR file for an enterprise bean.
6. Deploy the enterprise bean by doing one of the following:
    - To automatically deploy the enterprise bean from a JAR file, use the**CBDeployJar** tool. For more information, see Deploying an enterprise bean with the CBDeployJar tool.
    - To automatically deploy the enterprise bean from a J2EE EAR file, use the**CBDeployEar** tool. For more information, see Deploying an enterprise bean with the CBDeployEar tool.
    - To manually deploy the enterprise bean from a JAR file, do thefollowing:
        a. Use the **cbejb** command to deploy the enterprise bean. Formore information, see Using the cbejb tool to deploy enterprise beans.
        b. Build a data object (DO) implementation for use by the enterprise bean byusing Object Builder. For more information, see Building a data object during CMP entity bean deployment.
        c. Install the deployed enterprise bean and configure its EJB server(CB). For more information, see Installing an enterprise bean and configuring its EJB server (CB).
        d. Start the EJB server (CB). For more information see the ComponentBroker System Administration Guide.
        e. Bind the JNDI name of the enterprise bean into the JNDI namespace by usingthe **ejbbind** tool. (This step is not necessary on the AIX,Windows NT, Windows 2000, or Solaris platforms.) For more information,see Binding the JNDI name of an enterprise bean into the JNDI namespace.

    For more information on manual deployment, see Manually deploying an enterprise bean.

## Prerequisite software for the EJB server (CB)

**Note:**

Any items marked *PAO only* are needed only if you intend to use the**PAOToEJB** tool and need the CICS- or IMS-related support.

You must configure the tools provided with the EJB server (CB)environment; however, before you can configure the tools, you must ensurethat you have installed and configured the following prerequisite softwareproducts contained in the Enterprise Application Server:

- CB Server
- CB Tools (including the Object Builder, VisualAge Component Developmenttoolkit, samples, the Server SDK, and (*PAO only*) CICS and IMSApplication Adapter SDK
- (*PAO only*) CICS/IMS Application run time
- (*PAO only*) CICS/IMS Application client

## Setting the CLASSPATH environment variable in the EJB server (CB) environment

To do any of the tasks listed below, make sure that the classes.zipfile contained in the Java Development Kit is included in the CLASSPATHenvironment variable. In addition, make sure that the following filesare identified by the CLASSPATH environment variable to perform the associatedtask:

- Developing an enterprise bean or an EJB client: no additionalfiles.
- Deploying an EJB JAR file:
    - somojor.zip
    - The EJB JAR file being deployed and any JAR or ZIP files on which itdepends
- Running an EJB server (CB) managing an enterprise bean named*beanName*. These JAR files are automatically added to theCLASSPATH environment variable.
    - *beanName*S.jar
    - The EJB JAR file used to create *beanName*S.jar and anyJAR or ZIP files on which it depends
- Running a pure Java EJB client using an enterprise bean named*beanName*:
    - *beanName*C.jar
    - somojor.zip
- Running an EJB server (CB) that contains an enterprise bean named*clientBeanName* that accesses another enterprise bean named*beanName* as a client. These JAR files are automaticallyadded to the CLASSPATH environment variable.
    - *clientBeanName*S.jar
    - The EJB JAR file used to create *clientBeanName*S.jar andany JAR or ZIP files on which it depends
    - *beanName*C.jar

## Creating the components of an enterprise bean

If you use an ASCII text editor or a Java development tool that does notsupport enterprise bean development, you must create each of the componentsthat compose the enterprise bean you are creating. You must ensure thatthese components match the requirements of the EJB specification. Thesecomponents are described in Developing enterprise beans.

To manually develop a session bean, you must write the bean class, thebean's home interface, and the bean's remote interface. Tomanually develop an entity bean, you must write the bean class, thebean's primary key class, the bean's home interface, and thebean's remote interface.After you have properly coded these components, use the Java compiler tocreate the corresponding Java class files. For example, since thecomponents of the example Account bean

are stored in a specific directory, youcan compile the bean components by issuing the following command:

```
C:\MYBEANS\COM\IBM\EJS\DOC\ACCOUNT> javac *.java
```

This command assumes that the CLASSPATH environment variable contains allof the packages used by the Account bean.

## Creating finder logic in the EJB server (CB)

In the EJB server (CB), finder logic is contained in a finder helperclass. The enterprise bean deployer must implement the finder helperclass before deploying the enterprise bean and then specify the name of theclass with the -finderhelper option of the **cbejb** tool.

For each specialized finder method in the home interface (other than thefindByPrimaryKey method), the finder helper class must have a correspondingmethod with the same name and parameter types. When an EJB clientinvokes a specialized finder method, the generated CB home that implements theenterprise bean's home interface invokes the corresponding finder helpermethod to determine what to return to the EJB client.

The finder helper class must also have a constructor that takes a singleargument of type com.ibm.IManagedClient.IHome.When the CB home instantiates the finder helper class, the CB home passes areference to itself to the finder helper constructor. This allows thefinder helper to invoke methods on the CB home within the implementation ofthe finder helper methods, which is particularly useful when the CB home is anIQueryableIterableHome because the finder helper can narrow the IHome objectpassed to the constructor and invoke query service methods on the CBhome.

The names of the entity bean's container-managed fields are mapped tointerface definition language (IDL) attributes of the same name, except thatan underscore (_) is appended, in the business object (BO) interface, the CBkey class, and the CB copy helper class. These names are mapped exactlyto IDL attributes in the DO interface. For example, in the AccountBeanclass, the *accountId* variable is mapped to *accountId_* inthe BO interface, the CB key class, and the CB copy helper class, but ismapped to *accountId* in the DO interface.

This renaming is necessary, and relevant to finder helper classesimplemented by using the Component Broker Query Service, because the entitybean's remote interface can also have a property named*accountId* (of potentially a different type) that must also beexposed through the BO interface. If that is the case, then a queryover the BO attribute *accountId* is done in object space, whereas aquery over the BO attribute *accountId_* is done directly against theunderlying data source, which is typically more efficient.

If a home interface's specialized finder method returns a singleentity bean, then the corresponding method in the finder helper class mustreturn the java.lang.Object type. When invoked, thefinder helper method can return the EJB object, the CB key object, the entitybean's primary key object, or a CB managed object framework (MOFW)object. If the finder helper method returns a CB object or a primarykey object, the CB home determines the corresponding EJB object to return tothe EJB client.

If a home interface's specialized finder method returns ajava.util.Enumeration type, the corresponding finder helpermethod must also return java.util.Enumeration. Wheninvoked, the finder helper method can return an Enumeration of EJB objects, CBkey objects, CB MOFW objects, enterprise bean primary key objects, or aheterogeneous mix of one or more of the four. The CB home thenconstructs a serializable Enumeration object containing the corresponding EJBobjects, which is returned to the EJB client.

If a home interface's specialized finder method returns ajava.util.Collection type, the corresponding finder helpermethod must also return java.util.Collection. Wheninvoked, the finder helper method can return a Collection of EJB objects, CBkey objects, CB MOFW objects, enterprise bean primary key objects, or aheterogeneous mix of one or more of the four. The CB home thenconstructs a serializable Collection object containing the corresponding EJBobjects, which is returned to the EJB client.

An optional base class, namedcom.ibm.ejb.cb.runtime.FinderHelperBase, isprovided with the EJB server (CB) environment to assist in the development ofa finder helper class. This class encapsulates the Component BrokerQuery Service, so that the deployer does not need to write any CB-specificcode. The FinderHelperBase base class contains the methods listed in Table 1. These methods generally take an Object-OrientedStructured Query Language (OOSQL) predicate as a parameter and return anobject or an Enumeration or Collection of objects that meet the conditions ofthe query.

**Table 1. FinderHelperBase class methods**

| Method | Parameter | Return type | Notes |
|---|---|---|---|
| evaluate | OOSQL where clause | Enumeration | Desired objects instantiated immediately |
| extendedEvaluate | Full OOSQL statement | Enumeration | Desired objects instantiated immediately |
| lazyEvaluate | OOSQL where clause | Enumeration | Desired objects instantiated as needed |
| extendedLazyEvaluate | Full OOSQL statement | Enumeration | Desired objects instantiated as needed |
| singleEvaluate | OOSQL where clause | Object | Throws ObjectNotFoundException if not found |
| extendedSingleEvaluate | Full OOSQL statement | Object | Throws ObjectNotFoundException if not found |
| evaluateCollection | OOSQL where clause | Collection | Desired objects instantiated immediately |
| extendedEvaluateCollection | Full OOSQL statement | Collection | Desired objects instantiated immediately |
| lazyEvaluateCollection | OOSQL where clause | Collection | Desired objects instantiated as needed |
| extendedLazyEvaluateCollection | Full OOSQL statement | Collection | Desired objects instantiated as needed |

All of these methods throw a javax.ejb.FinderException if anyerrors occur. The finder helper class does not need to catch thisexception; instead, the class can pass it on to the EJB client.A utility class, named**com.ibm.ejb.cb.emit.cb.FinderHelperGenerator**(contained in the developEJB.jar file), is also provided to furtherassist the deployer in the development of a finder helper class. Thisutility takes the name of an entity bean's home interface and generates aJava source file containing a class that extendscom.ibm.ejb.cb.runtime.FinderHelperBase andthat contains skeleton methods for each specialized finder method in the homeinterface. In addition, each finder helper method contains a call toinvoke the appropriate FinderHelperBase method listed in Table 1.

By using **ejbfhgen**, the **FinderHelperGenerator** utility,the deployer can easily implement the finder helper class. You can usea batch file to run the utility. For example, to generate a finderhelper class for the example AccountHome interface, enter the followingcommand:

```
# ejbfhgen com.ibm.ejs.doc.account.AccountHome
```

This command generates the finder helper class shown in Figure 8.

**Figure 8. Code example: Generated AccountFinderHelper class for the EJB server (CB)**

```
...public class AccountFinderHelper extends FinderHelperBase {     ...
AccountFinderHelper(IManagedClient.IHome iHome) {          ...     }     public Enumeration
findLargeAccounts(float amount) {          return evaluate("replace with appropriate code");     } }
```

To enable the helper class for use in a deployed enterprise bean, thedeployer makes a few simple edits to the parameters of the evaluateinvocations. For example, for the AccountFinderHelper class, the`"replace with appropriate code"` String is replaced with`"balance_>" + amount` as shown in Figure 9. The generated finder helper class can be usedonly with an enterprise bean that is deployed to have a queryable home byusing the -queryable option of the **cbejb** tool.

**Figure 9. Code example: Completed AccountFinderHelper class for the EJB server (CB)**

```
...public class AccountFinderHelper extends FinderHelperBase {     ...
AccountFinderHelper(IManagedClient.IHome iHome) {          ...     }     public Enumeration
findLargeAccounts(float amount) {          return evaluate("balance_>" + amount);     } }
```

#### Using VisualAge for Java-style finder-helper interfaces

The VisualAge for Java finder-helper interfaces (described in Creating finder logic in the EJB server (AE)) support suffixes that map to the FinderHelperBase methodsas shown in Table 2.

**Table 2. FinderHelperBase method suffixes**

| Suffix | Return type | Method |
|---|---|---|
| CBWhereClause | Enumeration | evaluate |
| CBQueryString | Enumeration | extendedEvaluate |
| CBWhereClause | Collection | evaluateCollection |
| CBQueryString | Collection | extendedEvaluateCollection |
| CBWhereClause | Object | singleEvaluate |
| CBQueryString | Object | extendedSingleEvaluate |
| CBLazyWhereClause | Enumeration | lazyEvaluate |
| CBLazyQueryString | Enumeration | extendedLazyEvaluate |
| CBLazyWhereClause | Collection | lazyEvaluateCollection |
| CBLazyQueryString | Collection | extendedLazyEvaluateCollection |

VisualAge for Java will automatically create a CB finder-helper class whenyou export an EJB JAR file to CB with the CBWhereClause,CBQueryString,CBLazyWhereClause, or CBLazyQueryString specified in thefinder-helper interface.

Alternatively, you can manually create a CB finder-helper class by passingthe VisualAge for Java-style finder-helper interface as the second parameterto the **ejbfhgen** utility. For example, you could issue thefollowing command:

```
ejbfhgen com.ibm.ejs.doc.account.AccountHome      com.ibm.ejs.doc.account.AccountBeanFinderHelper
```

When this command is invoked with a VisualAge for Java-style finder-helperinterface as input, it fills in the OOSQL statements instead of emitting the`"replace with appropriate code"` string and compiles thecode. There is no need to manually edit the code when passing aVisualAge for Java-style finder-helper interface that contains all of theOOSQL strings. The deployer needs to add the compiled CB finder-helperclass to an EJB JAR file; alternatively, it can be packaged in a separateJAR file by using the **cbejb** tool with the -serverdepparameter.

#### Using lazy enumeration

The Enumeration returned by the evaluate method is called *eager*,because all the enterprise bean references that match the query are broughtinto memory and stored in the enumeration before being passed from the serverto the client. If the number of references returned by the query islarge, the deployer can use *lazy* enumeration; that is, itincrementally fetches more enterprise bean references only when the clientcalls the nextElement method on the Enumeration.

To use lazy enumeration, change the call to the evaluate method in theFinderHelper to a call to the lazyEvaluate method. A transaction mustalready be started before the home's finder method is called. Thecaller must not call the nextElement method on the Enumeration after thecompletion of the transaction.

At configuration time, the System Management End User Interface must beused to enable the settings for lazy Enumerations. Refer to Configuring systems management to enable lazy enumeration

## Creating an EJB JAR file for an enterprise bean

Once the bean components are built, the next step is package them into an EJBJAR file. The WebSphere Application Server **jetace** tool canbe used to create an EJB JAR file for one or more enterprise beans andgenerate a deployment descriptor file for each enterprise bean. Theresulting EJB JAR file contains each enterprise bean's class files anddeployment descriptor and an EJB-compliant manifest file.

**Note:**

The **jetace** tool can only be used to create JAR files that arecompatible with version 1.0 of the EJB specification. If youneed to create JAR files compatible with version 1.1, use theApplication Assembly tool. See Using the Application Assembly Tool.

Before you create an EJB JAR file for one or more enterprise beans, you mustdo *one* of the following:

- Place all of the components of each enterprise bean into a singledirectory.
- Create a standard JAR file that contains the class and interface files ofeach enterprise bean by using the Java Archiving tool (**jar**).The following command, when run from the root directory of the Accountbean's full package name, can be used to create the fileAccountIn.jar with a default manifest file:

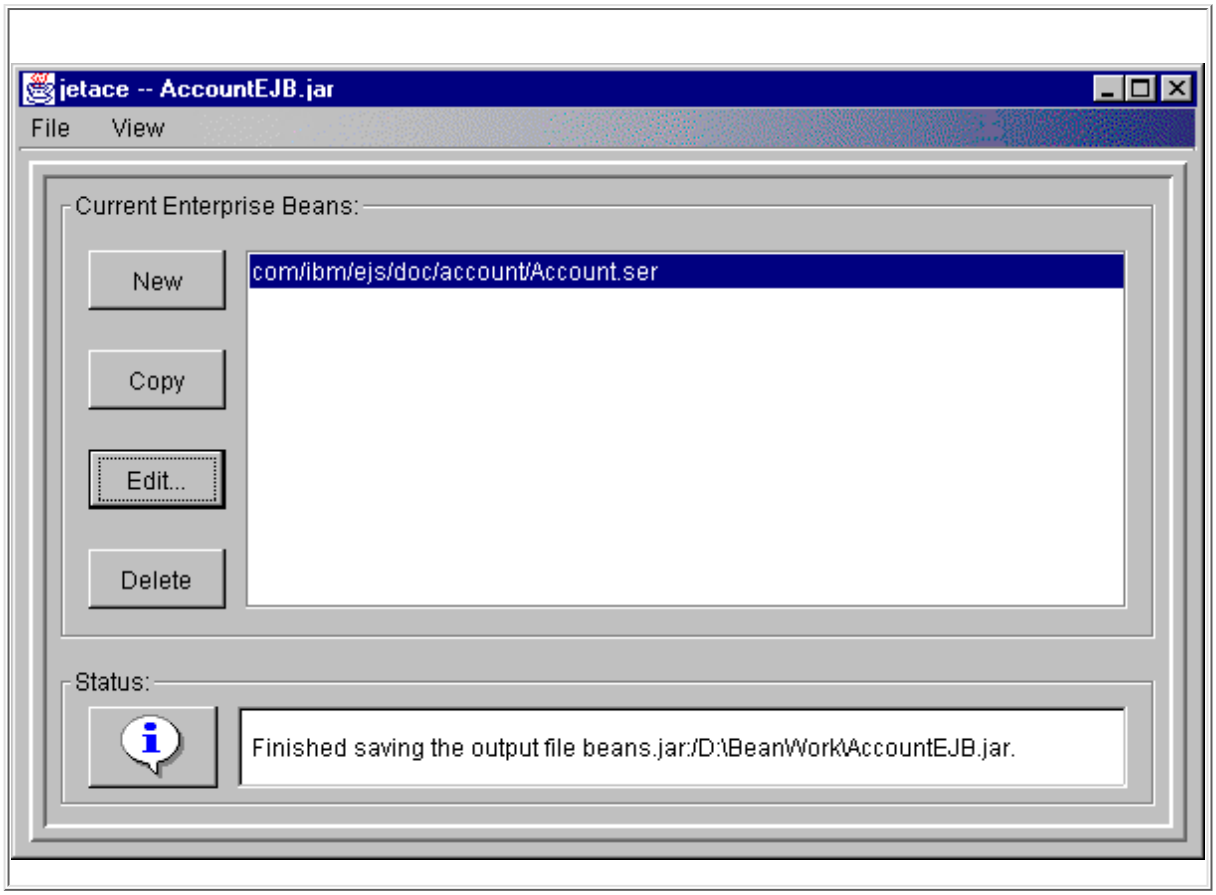
  ```
  C:\MYBEANS> jar cfv AccountIn.jar com\ibm\ejs\doc\account\*.class
  ```

- Create a standard ZIP file that contains the class and interface files ofeach enterprise bean by using a tool like WinZip(R).

#### Running the jetace tool

To run the **jetace** tool, type **jetace** on the commandline. The window shown in Figure 10 is displayed.

**Figure 10. The initial window of jetace tool**




To generate an EJB JAR file with the **jetace** tool, do thefollowing:

1. Click the **File->Load** item, and select the JAR or ZIP file orthe directory containing one or more enterprise beans. Use the**Browse** button to obtain the file or directory.

   **Note:**

   To specify the current directory as the input source, type an = (equalscharacter) in the **File Name** field of the browser window and click**Open**.

If you are creating a new EJB JAR file, click **New** and a defaultname for the deployment descriptor (for example, UNAMED_BEAN_1.ser)appears in the **Current Enterprise Beans** list box. (You canedit this name on any of the remaining tabbed pages of the **jetace**GUI by editing the **Deployed Name** field at the top of each tabbedpage. This field is described in Specifying the enterprise bean components and JNDI home name.)

If you are editing an existing EJB JAR file, the name of the deploymentdescriptor for each enterprise bean in the EJB JAR file is displayed in the**Current Enterprise Beans** list box, as shown in Figure 10.

- If you do not want to include a listed enterprise bean in the resultingEJB JAR file, highlight that enterprise bean's deployment descriptor andclick **Delete**. This action removes the deployment descriptorfrom the list box.
- If you want to create a duplicate of an enterprise bean, highlight itsdeployment descriptor and click **Copy**. This action adds a newdefault deployment descriptor to the list box. Copying can be useful ifyou want to create a deployment descriptor for one enterprise bean that issimilar to the deployment descriptor of the copied bean. You must thenedit the new deployment descriptor.

2. To create a new deployment descriptor or edit an existing one, highlightthe deployment descriptor and press the **Edit** button. Thisaction causes the **Basic** page to display. On this page, setor confirm the names of the deployment descriptor file, the enterprise beanclass, the home interface, and the remote interface and specify the JNDI nameof the enterprise bean. For information, see Specifying the enterprise bean components and JNDI home name.
3. Set the entity bean or session bean attributes for the enterprisebean's deployment descriptor on the **Entity** or**Session** page, respectively. For information on settingdeployment descriptor attributes for entity beans, see Setting the entity bean-specific attributes. For information on setting deployment descriptorattributes for session beans, see Setting the session bean-specific attributes.
4. Set the transaction attributes for the enterprise bean's deploymentdescriptor on the **Transactions** page. For information, see Setting transaction attributes.
5. Set the security attributes for the enterprise bean's deploymentdescriptor on the **Security** page. For information, see Setting security attributes.
6. Set any environment variables to be associated with the enterprise bean onthe **Environment** page. For information, see Setting environment variables for an enterprise bean.
7. Set any class dependencies to be associated with the enterprise bean onthe **Dependencies** page. For information, see Setting class dependencies for an enterprise bean.
8. After you have set the appropriate deployment descriptor attributes foreach enterprise bean, click **File->Save As** to create an EJB JARfile. (If desired, a ZIP file can be created instead of a JARfile.)

The **jetace** tool can also be used to read and generate an XMLversion of an enterprise bean's deployment descriptor. To read anXML file, click the **File->Read XML** item. To generate an XMLfile from an existing enterprise bean (after saving the output EJB JAR file)click the **File->Write XML** item.

The **jetace** tool can also be run from the command line to createan EJB JAR file. The syntax of this command follows, where*xmlFile* is the name of an XML file containing the enterprisebean's deployment descriptor:

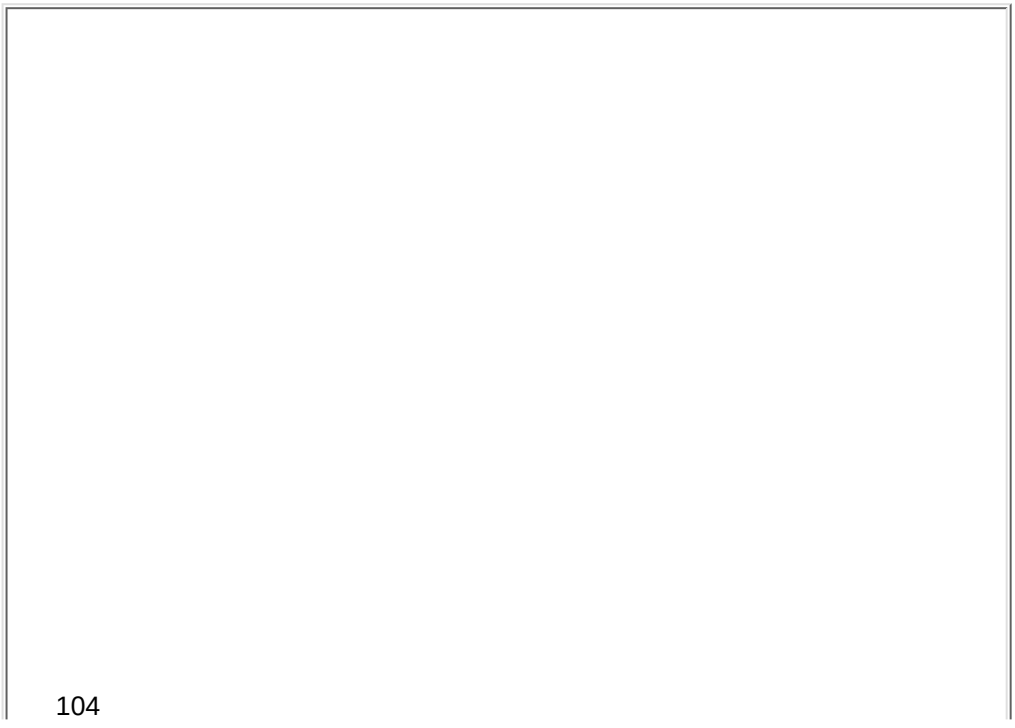
```
% jetace -f xmlFile
```

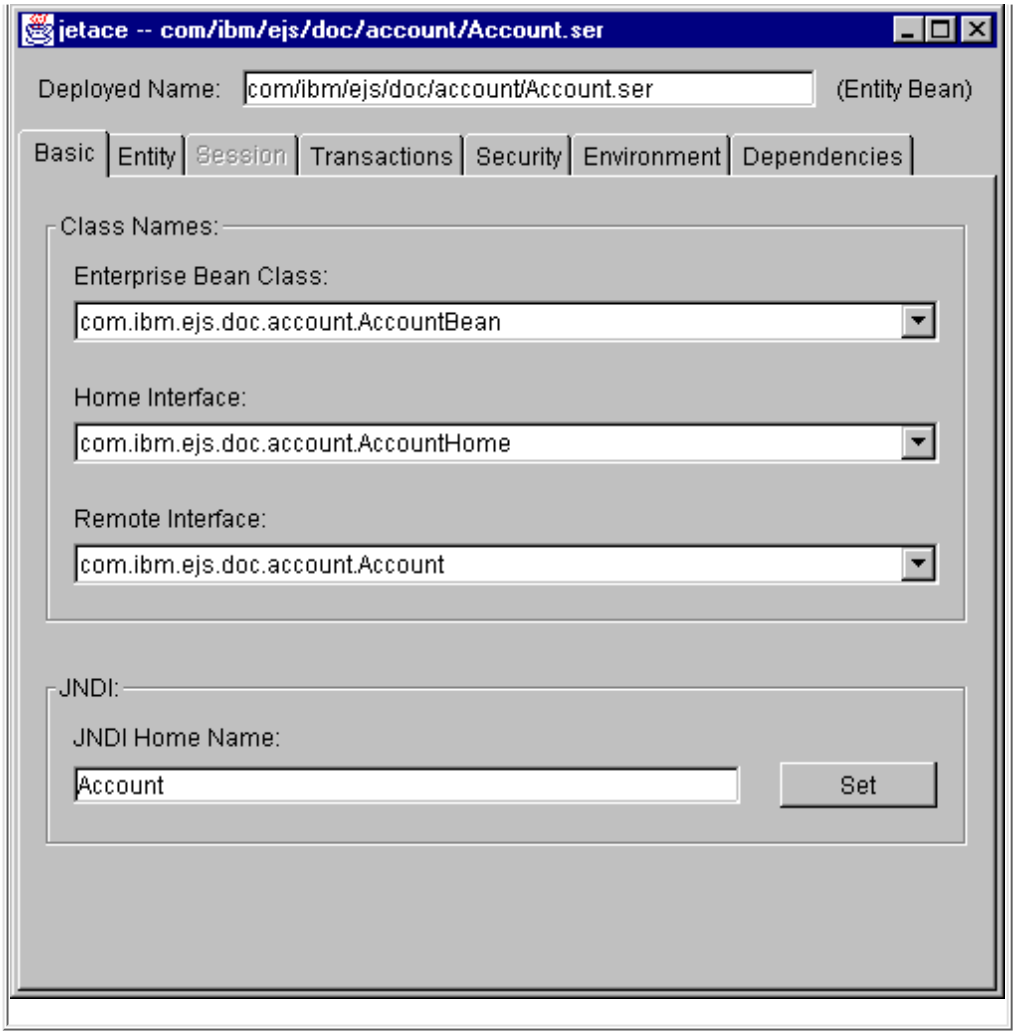
For more information on the syntax of the XML file required for thiscommand, see Appendix C, Using XML in enterprise beans (CB Only).

#### Specifying the enterprise bean components and JNDI home name

The **Basic** page is used to set the full pathname of the deploymentdescriptor file and the Java package name of the enterprise bean class, homeinterface, and remote interface and to set the enterprise bean's JNDIhome name. To access this page, which is shown in Figure 11, click the **Basic** tab.

**Figure 11. The Basic page of the jetace tool**





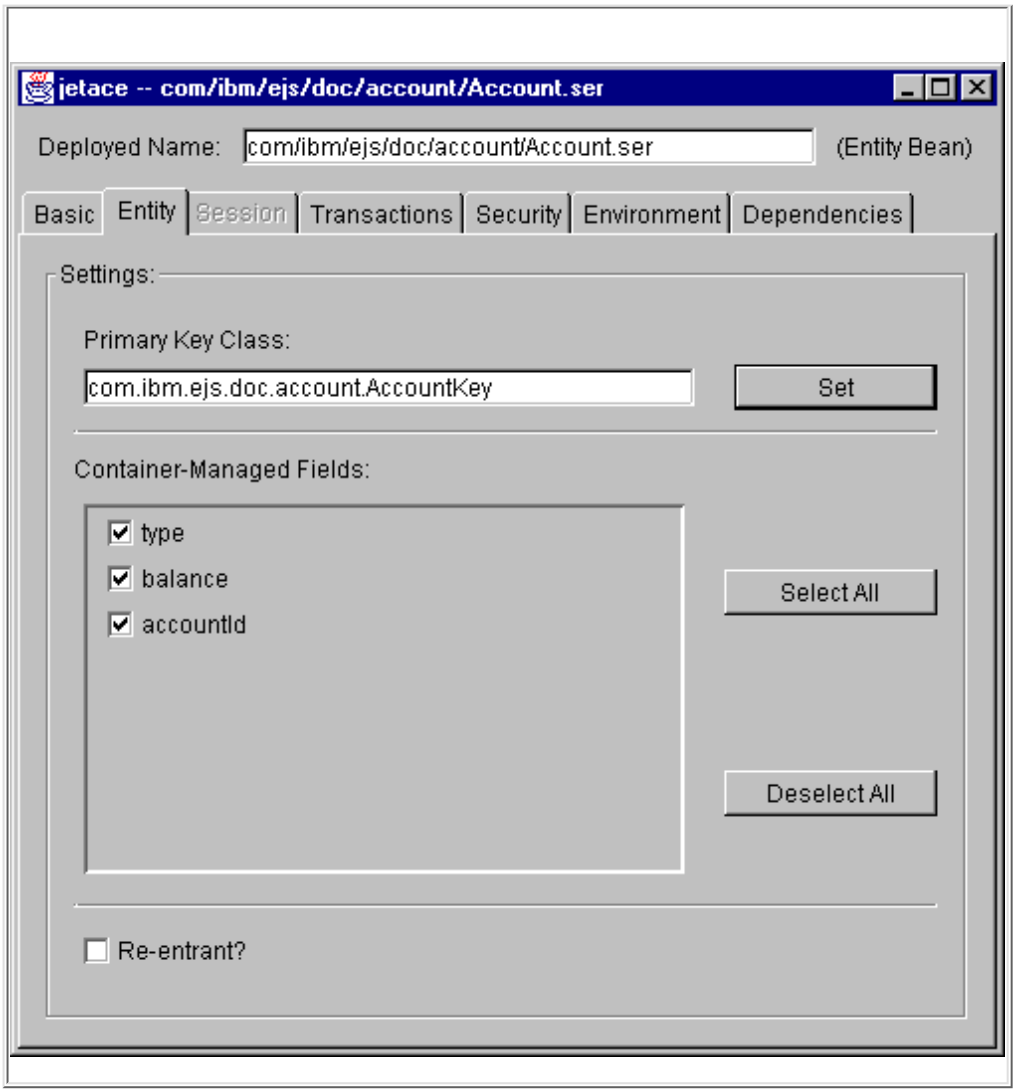

In the Basic page, you must select or confirm values for the followingfields:

- **Deployed Name**--The pathname of the deployment descriptorfile to be created. It is recommended that this directory name matchthe full package name of the enterprise bean class. For the Accountbean, the full name is`com/ibm/ejs/doc/account/Account.ser`.
- **Enterprise Bean Class**--Specify the full package name ofthe bean class. For the Account bean, the full name is`com.ibm.ejs.doc.account.AccountBean`.
- **Home Interface**--Specify the full package name of thebean's home interface. For the Account bean, the full name is`com.ibm.ejs.doc.account.AccountHome`.
- **Remote Interface**--Specify the full package name of thebean's remote interface. For the Account bean, the full name is`com.ibm.ejs.doc.account.Account`.
- **JNDI Home Name**--Specify the JNDI home name of thebean's home interface. This the name under which the enterprisebean's home interface is registered and therefore is the name that mustbe specified when an EJB client does a lookup of the home interface.For the Account bean, the JNDI home name is `Account`.

#### Setting the entity bean-specific attributes

To set the deployment descriptor attributes associated specifically with anentity bean, click the **Entity** tab in the **jetace** tool todisplay the **Entity** page shown in Figure 12. This tab is disabled if the highlighted enterprisebean in the initial **jetace** window is a session bean.

**Figure 12. The Entity page of the jetace tool**



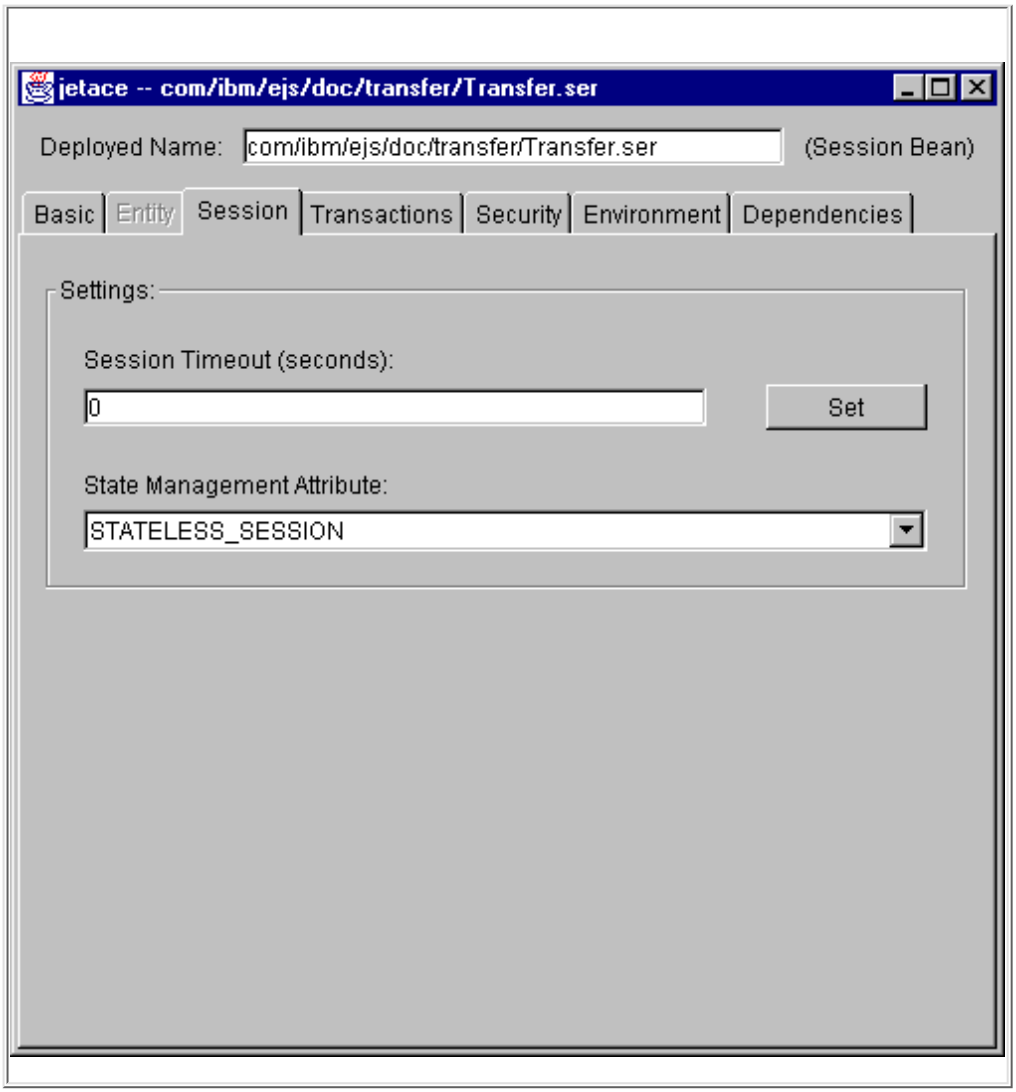

In the **Entity** page, you must select or confirm values for thefollowing fields:

- **Primary Key Class**--Specify the full package name of thebean's primary key class. For the example Account bean, the fullname iscom.ibm.ejs.doc.account.AccountKey.
- **Container-Managed Fields**--Check the check boxes of thevariables in the bean class for which the container needs to handlepersistence management. This is required for entity beans with CMPonly, and must *not* be done for entity beans with BMP. Forthe Account bean, the type, balance, and accountId variables are containermanaged, so each box is checked.
- **Re-entrant?**--Check this check box if the bean isreentrant. By default, an entity bean is not reentrant. If aninstance of a non-reentrant entity bean is executing a client request in atransaction context and it receives another request using the same transactioncontext, the EJB container throws the java.rmi.RemoteExceptionexception to the second request. Since a container cannot distinguishbetween a legal loopback call from another bean and an illegal concurrent callfrom another client or client thread, a client must take care to preventconcurrent calls to a reentrant bean. The example Account bean is*not* reentrant.

#### Setting the session bean-specific attributes

To set the deployment descriptor attributes associated specifically with asession bean, click the **Session** tab in the **jetace** toolto display the **Session** page shown in Figure 13. This tab is disabled if the highlighted enterprisebean in the initial **jetace** window is an entity bean.

**Figure 13. The Session page of the jetace tool**




On the **Session** page, you must select or confirm values for thefollowing fields:

- **Session Timeout (seconds)**--Specify the idle timeout valuefor this bean in seconds; a `0` (zero) indicates that idle beaninstances timeout after the maximum allowable timeout period haselapsed. For the Transfer bean, the value is left at `0` toindicate that the default timeout is used.

  **Note:**

  In the EJB server (CB) environment, this attribute is not used.
- **State Management Attribute**--Specify whether the bean isstateless or stateful. The example Transfer bean is`STATELESS_SESSION`. For more information, see Stateless versus stateful session beans.

#### Setting transaction attributes

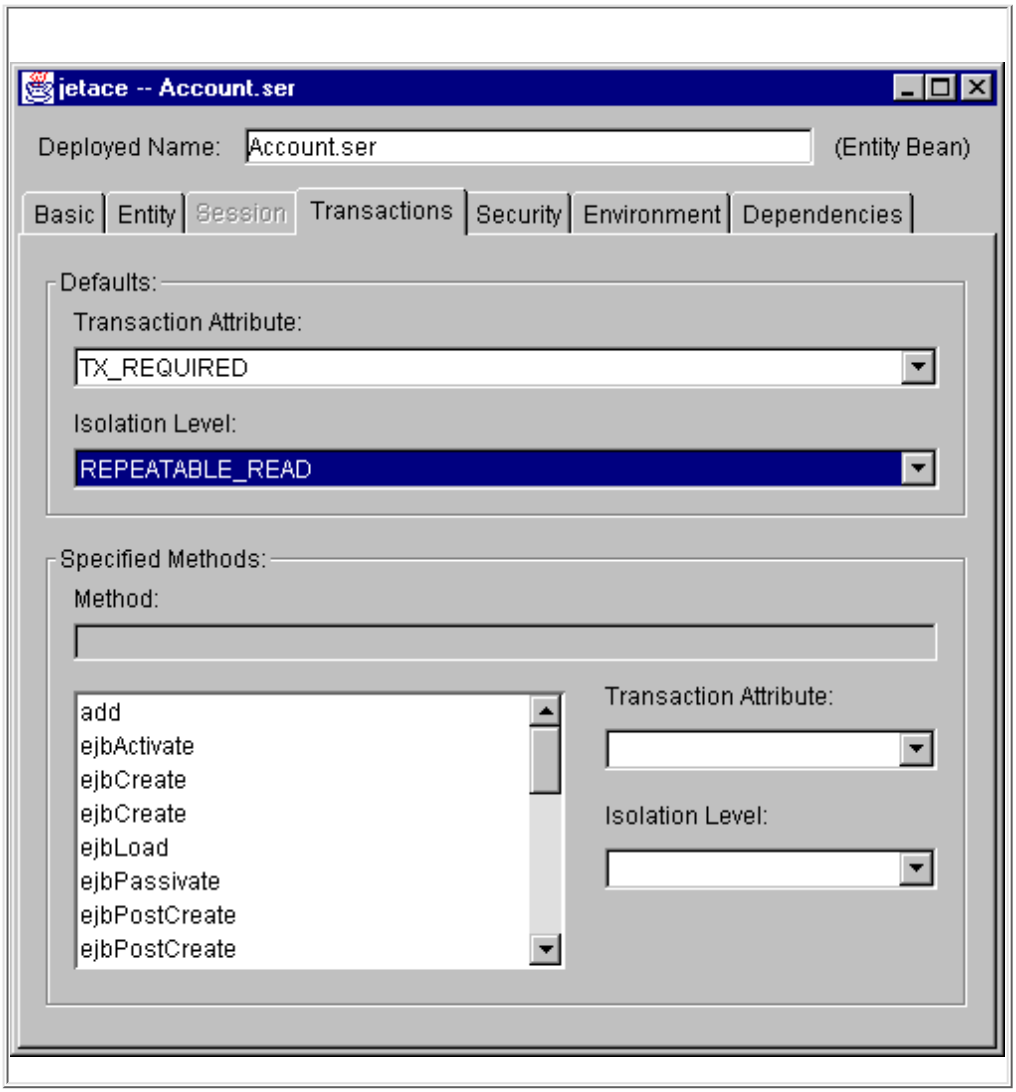
The **Transactions** page is used to set the transaction andtransaction isolation level attributes for all of the methods in an enterprisebean and for individual methods in an enterprise bean. If an attributeis set for an individual method, that attribute overrides the defaultattribute value set for the enterprise bean as a whole.

**Note:**

In the EJB server (CB), the transactional attribute can be set only for thebean as a whole; the transaction attribute cannot be set on individualmethods in a bean.

To access the **Transaction** page, click the**Transactions** tab in the **jetace** tool. Figure 14 shows an example of this page.

**Figure 14. The Transactions page of the jetace tool**




On the **Transactions** page, you must select or confirm values forthe following fields in the **Defaults** group box:

- **Transaction Attribute**--Set a value for the transactionattribute. The values for this attribute are described in Enabling transactions and security in enterprise beans. For the Account bean, the value`TX_MANDATORY` is used because the methods in this bean must beassociated with an existing transaction when invoked; as a result, theTransfer bean must use the value that begins a new transaction or passes on anexisting one.
- **Isolation Level**--Set a value for the transaction isolationlevel attribute. The values for this attribute are described in Enabling transactions and security in enterprise beans. For the Account bean, the value`REPEATABLE_READ` is used.

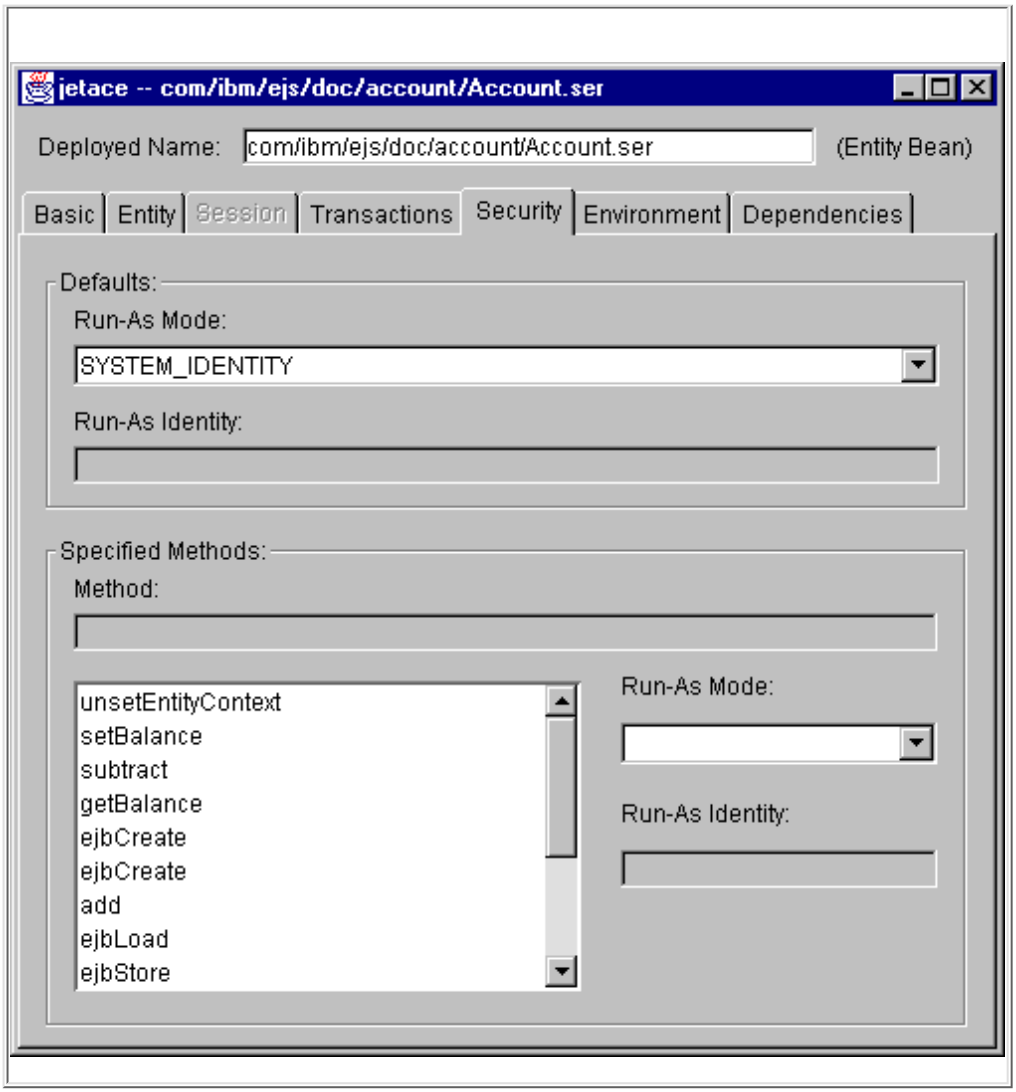
If necessary, you can also set these attributes on individual methods byhighlighting the appropriate method and setting one or both of the attributesin the **Specified Methods** group box.

#### Setting security attributes

The **Security** page is used to set the security attributes for all ofthe methods in an enterprise bean and for individual methods in an enterprisebean. If an attribute is set for an individual method, that attributeoverrides the default attribute value set for the enterprise bean as awhole.

To access the **Security** page, click the **Security** tabin the **jetace** tool. Figure 15 shows an example of this page.

**Figure 15. The Security page of the jetace tool**



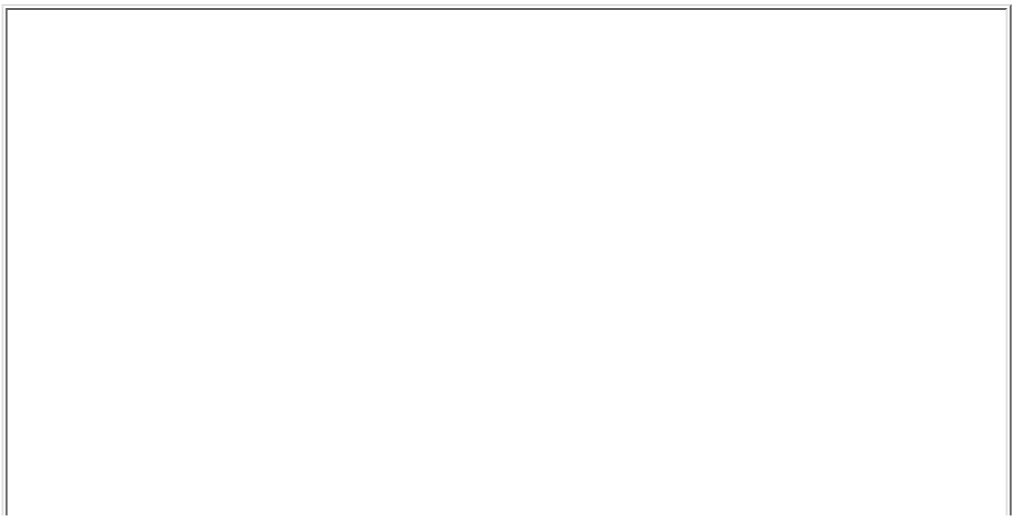

On the **Security** page, you must select or confirm values for the**Run-As Mode** field in the **Defaults** group box. Thisfield must be set to one of the values described in Setting the security attribute in the deployment descriptor. The *run-as identity* attribute is not usedby the EJB server (CB environment), so you cannot set the value for thecorresponding field in the **jetace** tool.

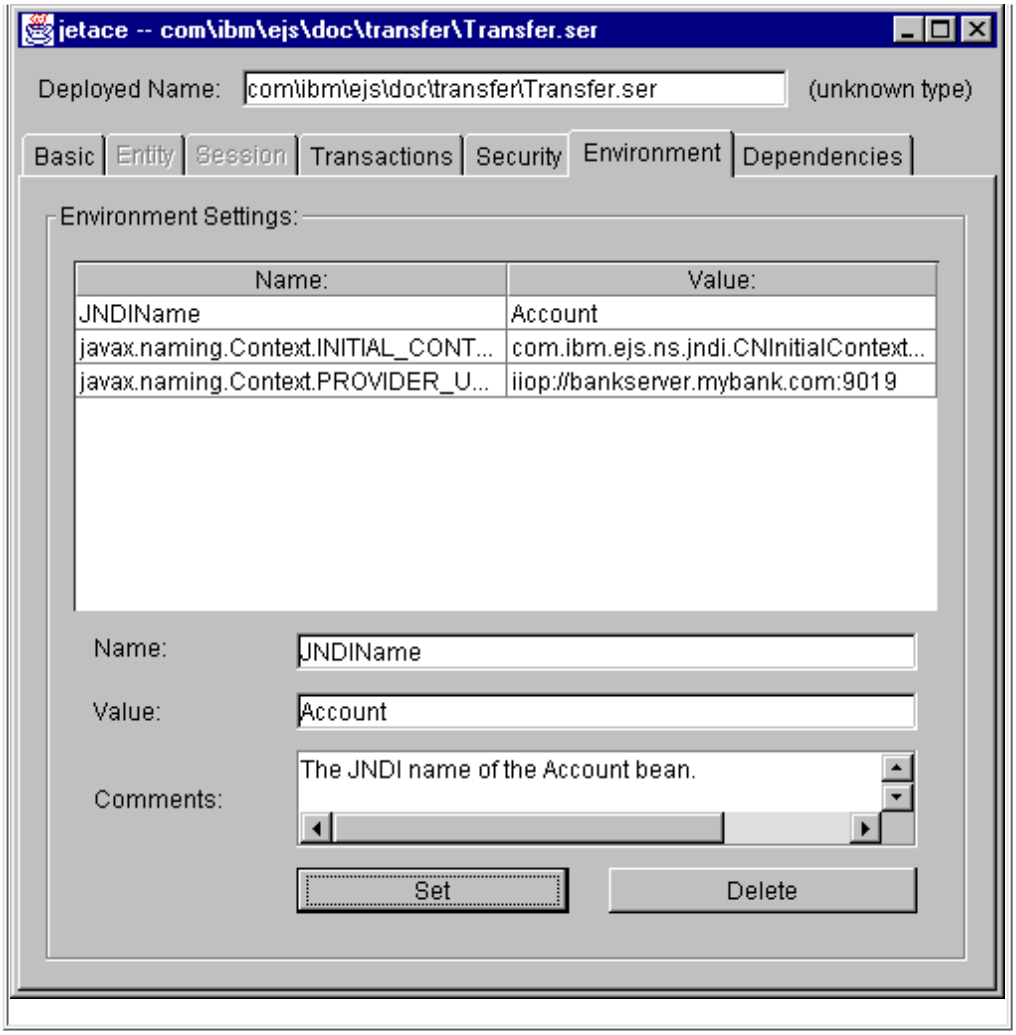
If necessary, you can also set the *run-as mode* attribute onindividual methods by highlighting the appropriate method and setting theattribute in the **Specified Methods** group box.

#### Setting environment variables for an enterprise bean

The **Environment** page is used to associate environment variables(and their corresponding values) with an enterprise bean. To access the**Environment** page, click the **Environment** tab in the**jetace** tool. Figure 16 shows an example of this page.

**Figure 16. The Environment page of the jetace tool**






To set an environment variable to its value, specify the environmentvariable name in the **Name** field and specify the environmentvariables value in the **Value** field. If desired, use the**Comment** field to further identify the environment variable.Press the **Set** button to set the value. To delete anenvironment variable, highlight the variable in the **EnvironmentSettings** window and press the **Delete** button.

For the example Transfer bean, the following environment variables arerequired:

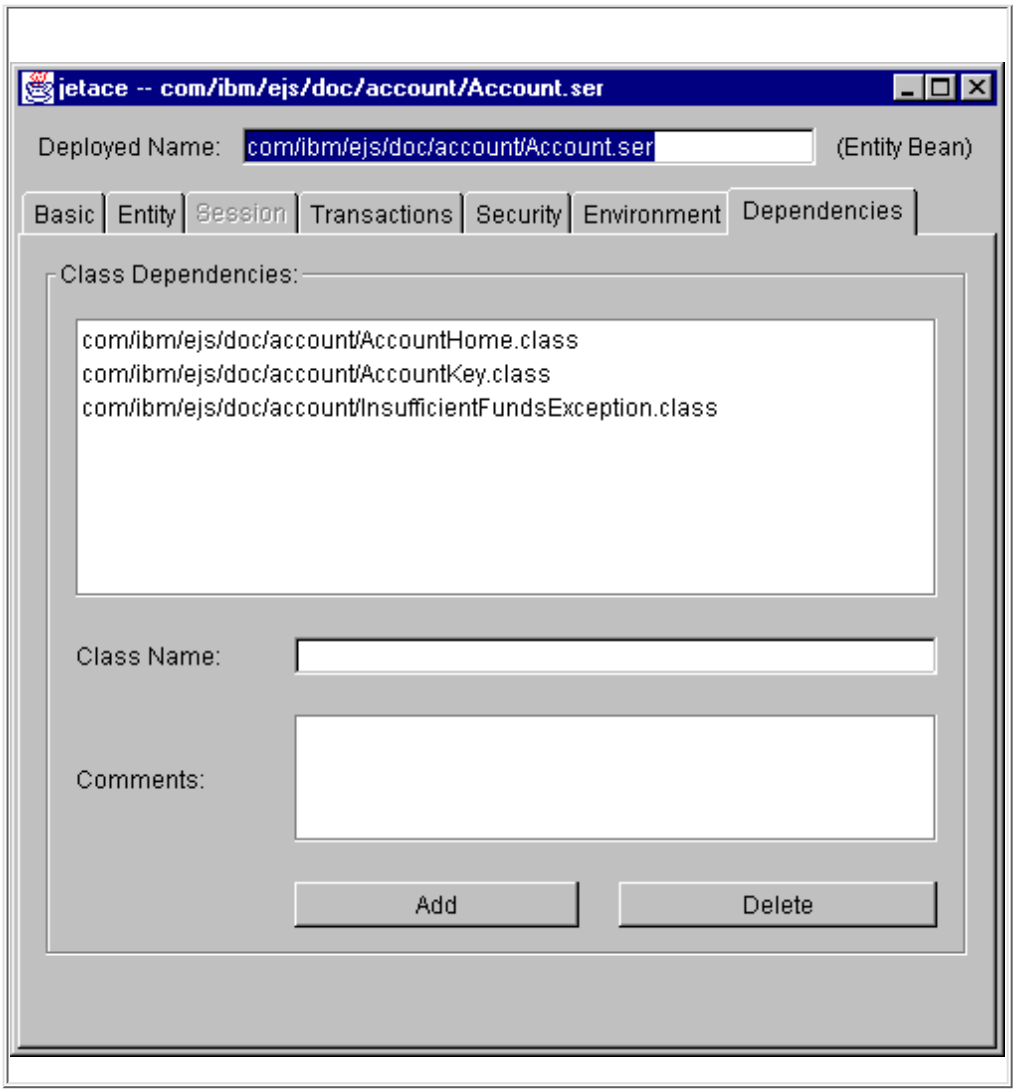
- JNDIName--The JNDI name of the Account bean, which is accessed by theTransfer bean. For more information, see Figure 11.
- javax.naming.Context.INITIAL_CONTEXT_FACTORY--Thename of the initial context factory used by the Transfer bean to look up theJNDI name of the Account bean
- javax.naming.Context.PROVIDER_URL--The locationof the naming service used by the Transfer bean to look up the JNDI name ofthe Account bean.

For more information on how these environment variables are used by theTransfer bean, see Implementing the ejbCreate methods.

#### Setting class dependencies for an enterprise bean

The **Dependencies** page is used to specify classes on which theenterprise bean depends. To access the **Dependencies** page,click the **Dependencies** tab in the **jetace** tool. Figure 17 shows an example of this page.

**Figure 17. The Dependencies page of the jetace tool**




Generally, the **jetace** tool discovers class dependenciesautomatically and sets them here. If there are other class dependenciesrequired by an enterprise bean, you must set them here by entering thefully-qualified Java class name in the **Classname** field. Ifdesired, use the **Comment** field to further identify thedependency. Press the **Add** button to set the value. Toremove a dependency, highlight it in the **Class Dependencies** windowand press the **Delete** button.

For the example Account bean, the **jetace** tool set thedependencies shown in Figure 17.

### Deploying an enterprise bean with the CBDeployJar tool

The **CBDeployJar** tool automates the tasks associated withdeploying an enterprise bean. It can be used to do the following:

- Deploy enterprise beans from JAR files
- Verify whether enterprise beans have been deployed from a JAR file
- Undeploy enterprise beans associated with a JAR file

The **CBDeployJar** tool can be run on JAR files that are compatiblewith both version 1.0 and version 1.1 of the EJBspecification. It can be used to deploy the following types ofenterprise beans:

- Session beans
- Entity beans with BMP
- Entity beans with CMP that use top-down mapping or have mappinginformation from VisualAge for Java

It cannot be used to deploy entity beans with CMP that usemeet-in-the-middle mapping and were not created using VisualAge forJava. These enterprise beans must be manually deployed as described in Manually deploying an enterprise bean.

When it deploys an enterprise bean from a JAR file, the**CBDeployJar** tool performs the following tasks:

1. If the JAR file is compatible with version 1.1 of the EJBspecification, it parses the XML of the version 1.1 deploymentdescriptor and generates a new JAR file with version 1.0-styleserialized deployment descriptors. (This is necessary because otherComponent Broker EJB tools only work with version 1.0 JARfiles.) It also registers all EJB 1.1 deployment descriptorenvironment variables in the JNDI namespace underjava:comp/env/*environVarName*, where *environVarName*is the name of the environment variable.
2. It runs the **cbejb** tool on the JAR file, using any optionsspecified by the user.
3. It runs the **make** command for the platform, using any optionsspecified by the user.

4. It maps the persistent fields in entity beans with CMP to databasetables.
5. It configures and starts a Component Broker EJB server by running a seriesof **wscmd** commands that load the application family into ComponentBroker systems management; create a new management zone, configuration,and EJB server; configure the deployed enterprise beans onto the EJBserver; and start the EJB server.
6. For enterprise beans written to version 1.1 of the EJBspecification, it registers references to these beans in the appropriate placein the JNDI namespace under java:comp/env/ejb. (This is necessaryto prevent naming collisions between enterprise beans.)

The syntax of the **CBDeployJar** command is as follows:

```
CBDeployJar ejb-jarFile hostname [-cbejb options] [-make options] [-noTables]      [-prepJarOnly]
[-cbejbOnly] [-makeOnly]CBDeployJar ejb-jarFile hostname  -isDeployedCBDeployJar ejb-jarFile
hostname  -undeploy
```

where:

- ejb-jarFile -- The name of the JAR file (required).This must be the first parameter.
- hostname -- The fully-qualified host name of the machinewhere the enterprise beans are being deployed (required). This must bethe second parameter.
- -cbejb options -- Specifies the desired options for the**cbejb** command, which is run by the **CBDeployJar** tool aspart of the deployment process. If this flag is not set, thecommand's default options are used. For a complete list of**cbejb** command-line options, see Using the cbejb tool to deploy enterprise beans.

**Note:**

Use double quotes ("") for options passed to the -cbejb and -make flags thatcontain spaces.

- -make options -- Specifies the desired options for the**make** command for the platform, which is run by the**CBDeployJar** tool as part of the deployment process. If thisflag is not set, the command's default options are used. For acomplete list of **make** options, see the documentation for yourcompiler.
- -noTables -- Prevents the **CBDeployJar** tool from creatingtables for persistent fields in entity beans with CMP. You must specifythis flag if you are using entity beans with CMP that are backed by a databaseother than DB2. (The **CBDeployJar** tool only creates DB2database tables.)
- -prepJarOnly -- Stops the process after converting a version1.1-compatible JAR file to the version 1.0 format (step1).
- -cbejbOnly -- Stops the process after running the **cbejb**tool (step 2).
- -makeOnly -- Stops the process after running the **make**command (step 3).
- -isDeployed -- Verifies whether a specific JAR file has beendeployed. This option can be specified only with theejb-jarFile and hostname parameters.
- -undeploy -- Undeploys a JAR file that had previously been deployedwith the **CBDeployJar** tool. This option can be specified onlywith the ejb-jarFile and hostname parameters. The-undeploy option removes all of the files generated by the **cbejb**and **make** commands, deletes the Data Object implementations forentity beans with CMP, deletes the server configuration and associatedinformation, stops the EJB server, and deletes any references to EJB1.1-compatible enterprise beans from the JNDI namespace.

The following are examples of using the **CBDeployJar**command:

```
CBDeployJar EJBsavingsAccount.jar test.netbank.ibm.comCBDeployJar EJBcalculator.jar trident.ibm.com
-make IVB_COMBINE_SOURCE=0CBDeployJar EJBportfolio.jar bringup.ibm.com -cbejb "-dbname
Investors"CBDeployJar EJBhello.jar tasmania.ibm.com -noTablesCBDeployJar EJBtest.jar trip.ibm.com
-isDeployedCBDeployJar EJBtest.jar trip.ibm.com -undeploy
```

## Deploying an enterprise bean with the CBDeployEar tool

The **CBDeployEar** tool automatically deploys enterprise beans fromJAR files encapsulated in J2EE EAR files. This tool extracts a JAR filefrom the specified EAR file, then runs the **CBDeployJar** tool on theextracted file to deploy the enterprise bean.

The syntax of the **CBDeployEar** command is as follows:

```
CBDeployEar earFile hostname [-cbejb options] [-make options] [-noTables] [-prepJarOnly]
[-cbejbOnly] [-makeOnly] [-bindEJBRefs]CBDeployEar earFile hostname  -isDeployedCBDeployEar earFile
hostname  -undeploy
```

where:

- earFile -- The name of the J2EE EAR file (required).This must be the first parameter.
- hostname -- The fully-qualified host name of the machinewhere the enterprise beans are being deployed (required). This must bethe second parameter.
- -cbejb options -- Specifies options for the **cbejb**command, which is run when the **CBDeployEar** tool calls the**CBDeployJar** tool. See Deploying an enterprise bean with the CBDeployJar tool for more information.
- -make options -- Specifies options for the **make**command, which is run when the **CBDeployEar** tool calls the**CBDeployJar** tool. See Deploying an enterprise bean with the CBDeployJar tool for more information.
- -noTables -- Stops the tool from creating tables for persistentfields in entity beans with CMP. You must specify this flag if you areusing entity beans with CMP that are backed by a database other thanDB2.
- -prepJarOnly -- Stops the process after converting a version1.1-compatible JAR file to the version 1.0 format.
- -cbejbOnly -- Stops the process after running the **cbejb**tool.
- -makeOnly -- Stops the process after running the **make**command.

- -bindEJBRefs -- Binds references to EJB 1.1-compatibleenterprise beans into the JNDI namespace. This option is specified bydefault. However, there are situations when it is convenient to skipthe **cbejb** and **make** steps and perform the JNDI bindingstep when running the **CBDeployEar** tool-- for example, if theEJB server was not started when you first ran the **CBDeployEar** tooland you want to save time when running the tool again.
- -isDeployed -- Verifies whether a JAR file has been deployed from aspecific EAR file. This option can be specified only with theearFile and hostname parameters.
- -undeploy -- Undeploys a JAR file that had previously been deployedfrom an EAR file with the **CBDeployEar** tool. This option canbe specified only with the earFile and hostnameparameters.

The following are examples of using the **CBDeployEar**command:

```
CBDeployEar EJB11Big3.ear greenland.ibm.comCBDeployEar EJB11Big3.ear greenland.ibm.com
-bindEJBRefsCBDeployEar EJB11Big3.ear greenland.ibm.com -isDeployedCBDeployEar EJB11Big3.ear
greenland.ibm.com -undeploy
```

## Manually deploying an enterprise bean

You can manually deploy JAR files that contain any type of enterprise bean,regardless of which tool was used to create the files. The followingsteps summarize the tasks that you must complete to manually deploy enterprisebeans onto a Component Broker EJB server:

1. Use the **cbejb** command to deploy the enterprise bean.
2. Build a data object (DO) implementation for use by the enterprise bean byusing Object Builder (This step is part of the deployment process).
3. Install the deployed enterprise bean and configure its EJB server(CB).
4. Start the EJB server (CB) as described in the Component Broker System Administration Guide.
5. Bind the JNDI name of the enterprise bean into the JNDI namespace by usingthe **ejbbind** tool. (This step is not necessary on the AIX,Windows NT, Windows 2000 or Solaris platforms.)

This section describes how to perform steps 1, 2, 3 and 5.

#### Using the cbejb tool to deploy enterprise beans

During deployment, a deployed JAR file is generated from an EJB JARfile. Use the **cbejb** tool to deploy enterprise beans in theEJB server (CB) environment. The deployed JAR file contains classesrequired by the EJB server. The **cbejb** tool also generatesthe data definition language (DDL) file used during installation of theenterprise bean into the EJB server (CB).

If you want to use an enterprise bean on a different machine from the oneon which it was developed (and on which you ran **cbejb**), follow theguidelines for installing applications in the Component Broker documententitled System Administration Guide. If anenterprise bean uses additional files (such as other JAR files) that need tobe copied with the enterprise bean, specify these files in the propertiesnotebook of the application (*not* the family).

**Note:**

> The **cbejb** tool can only be used to deploy JAR files that arecompatible with version 1.0 of the EJB specification. Tomanually deploy a version 1.1-compatible JAR file, you must firstrun the **CBDeployJar** tool with the -prepJarOnly option to convertthe JAR file to the version 1.0 format. See Deploying an enterprise bean with the CBDeployJar tool for more information.

The **cbejb** tool has the following syntax:

```
cbejb ejb-jarFile [-rsp responseFile][-ob projDir] [-nm] [-ng] [-nc] [-cc] [-bean beanNames]
[-platform [NT | AIX | OS390 | Solaris | HP]][-guisg] [-usecurdopo] [-nousraction] [-dllname DLLName
beanName][-polymorphichome [beanNames]]  [-queryable [beanNames]][-dbname DBName
[beanName]][-cacheddb2v52 | -cacheddb2v61 | -db2v61 |-oracle | -informix |-jdbcaa [beanNames]][-hod
| -eci | -appc | -exci | -otma | -ccf [beanNames]] [-family familyName [beanNames]] [-finderhelper
finderHelperClassName [beanNames]] [-usewstringindo [beanNames]]  [-workloadmanaged
[beanNames]][-clientdep deployed-jarFile [beanNames]] [-serverdep deployed-jarFile
[beanNames]][-sentinel [JavaPrimitiveObjectType=]sentinelValue
[beanNames[+CMFieldNames]][-strbehavior [strip | corba] [beanNames[+CMFieldNames]]
```

The *ejb-jarFile* parameter is required; it must be the firstargument and it must specify a valid EJB JAR file. If the -ob option isused, it must come second on the command line. The other options can bespecified in any order. The *beanNames* argument is a list ofone or more fully qualified enterprise bean names delimited by colons(:) (for example,com.ibm.ejs.doc.transfer.Transfer:com.ibm.ejs.doc.account.Account).For the enterprise bean name, specify either the bean's remote interfacename or the name of its deployment descriptor. If the*beanNames* argument is not specified for a particular option, thenthe effect of that option is applied to all enterprise beans in the EJB JARfile for which the option is valid.

**Note:**

> The relative file name of the JAR files specified by the*ejb-jarFile* variable and by the two *deployed-jarFile*variables must be different from each other. JAR file names that havethe same relative file names but different paths are not valid.

The rest of the command parameters are optional and can be specified in anyorder. For explanation purposes, the options can be grouped by functioninto three general categories:

- Deployment options, which govern the generation and compilation ofcode.
- Storage options, which govern persistent storage.
- Execution options, which govern the run time environment.

The -rsp option does not fit into these categories. This optionallows you to create a file containing some or all of the other options andtheir values (except the *ejb-jarFile* parameter). You canthen submit the file to the **cbejb** command. This allows thecommon setting to be saved and makes commands easier to issue.

- Deployment options
  - -ob *projDir* -- Specifies the relative or full path of theproject directory in which the generated files are stored. If thisoption is not specified, the current working directory is used as the projectdirectory.
  - Compilation modifiers -- By default, the **cbejb** tool doesthe following for each enterprise bean contained in the EJB JAR file:
    1. Generate and import XML.
    2. Generate code--Creates a DDL file, makefile, and other source filesfor each enterprise bean contained in the EJB JAR file. These files areplaced in the specified project directory.
    3. Compile and link--Invokes the generated makefile to compile anapplication. Each application file is placed in the specified projectdirectory. While the Dynamic Link Libraries (DLLs) are being linked,numerous duplicate symbol warnings appear; these warnings are harmlessand can be ignored.

    The following command options modify the default compilationbehavior:
    - -nm -- Suppresses the XML-processing step.
    - -ng -- Suppresses the code-generation step.
    - -nc -- Suppresses the compilation-and-linking step.
    - -cc -- Removes previously compiled and linked code by invoking thegenerated makefile to remove non-source files. This option must be usedif you specify either of these combinations:
      - -ng -nc
      - -nm -ng -nc
  - -bean *beanNames* -- Identifies the enterprise beans in theEJB JAR file to be deployed. By default, all enterprise beans in theEJB JAR file are deployed. To deploy multiple enterprise beans, delimitthe bean names with a `:` (colon). For example,`Account:Transfer`.
  - -platform -- Specifies the platform for which to generatecode. This also sets the deployment platform in the Object Buildertool, but it does not set the platform for viewing, generating, or applyingdevelopment constraints. You must set these manually by using thechoices on the **Platform** menu.
  - -guisg -- Directs the tool to present the Object Builder graphicaluser interface (GUI), which enables the tool to collect options from the userrather than from the command line.
  - -usecurdopo -- Directs the tool to use the current mapping betweenthe data object and the persistent object in the existing model rather thanbringing up the Object Builder interface to build a mapping. Use thisoption when redeploying beans for which a satisfactory mapping alreadyexists. The deployment will proceed automatically.

    When you first deploy CMP entity beans, you must *not* use thisoption. The tool will then build the default mapping between the dataand persistent objects and, if you specify the -guisg option, launch theObject Builder interface.
  - -nousraction -- Directs the tool to use only the information on thecommand line after building the mapping between data objects and persistentobjects. Otherwise, if you have also specified the -guisg option, thetool prompts you for the next action.
  - -polymorphichome -- Specifies the beans that use polymorphic homeinterfaces.
  - -queryable -- Directs the tool to generate a queryable CB homeobject. This option can be used only for entity beans with CMP thatstore their persistent data in a relational database. This option mustbe used if the finder helper class, which is used to implement the findermethods in a CMP entity bean, uses the CB query service. This optionmust *not* be used if an entity bean uses CICS or IMS to store itspersistent data.

    By default, the interface definition language (IDL) interface of anenterprise bean's CB home extends the IManagedClient::IHomeclass, and the home implementation extends theIManagedAdvancedServer::ISpecializedHome class. An IDLinterface of a queryable home extends theIManagedAdvancedClient::IQueryableIterableHome class, and the homeimplementation extends theIManagedAdvancedServer::ISpecializedQueryableIterableHomeclass.

    In addition, the generated BO interface is marked as queryable. Forqueryable homes, the EJB client programming model remains unchanged;however, a Common Object Request Broker Architecture (CORBA) EJB client cantreat the EJB home as anIManagedAdvancedClient::IQueryableIterableHome object.

    For more information on queryable homes, see the Advanced Programming Guide.
- Storage options
  - -dbname *DBName* -- Specifies the name of the database forbeans with CMP.
  - Database choices--The default database for persistent storage ofcontainer-managed beans is DB2 version 5.2 with embedded SQL.You can override this default by using:
    - -cacheddb2v52 -- Identifies entity beans with CMP that require DB2version 5.2 used with the Cache Service to store persistentdata.
    - -cacheddb2v61 -- Identifies entity beans with CMP that require DB2version 6.1 used with the Cache Service to store persistentdata.
    - -db2v61 -- Identifies entity beans with CMP that require DB2 version6.1 used with embedded SQL to store persistent data.
    - -oracle -- Identifies entity beans with CMP that require Oracle tostore persistent data. If you specify this option, you must also usethe -queryable option.
    - -informix -- Identifies entity beans with CMP that require Informixto store persistent data. A given transaction cannot access more thanone Informix database from a CB server. To access two Informixdatabases in one transaction, you must access each from a different CBserver. If you specify this option, you must also use the -queryableoption.
    - -jdbcaa -- Identifies entity beans with BMP that require JDBC tostore persistent data. This option enables the beans to joindistributed transactions by allowing the bean implementation to connect to theTransaction Service. Beans with BMP that do not use this option willhandle transactions in an implementation-dependent manner.
  - -hod -- Identifies entity beans with CMP that use Host-on Demand(HOD) to store persistent data. These beans will use the SessionService. This option must *not* be used for enterprise beansgenerated from the **PAOToEJB** tool.

- -eci -- Identifies entity beans with CMP that use the external callinterface (ECI) to store persistent data. These beans will use theSession Service. This option must *not* be used for enterprisebeans generated from the **PAOToEJB** tool.
- -appc -- Identifies entity beans with CMP that use advancedprogram-to-program communications (APPC) to store persistent data.These beans will use the Transaction Service. This option must*not* be used for enterprise beans generated from the**PAOToEJB** tool.
- -exci -- Identifies entity beans with CMP that use the EXCI to storepersistent data. These beans will use the Transaction Service.This option must *not* be used for enterprise beans generated fromthe **PAOToEJB** tool.
- -otma -- Identifies entity beans with CMP that use the OTMA to storepersistent data. These beans will use the Transaction Service.This option must *not* be used for enterprise beans generated fromthe **PAOToEJB** tool.
- -ccf -- Identifies entity beans with CMP that use the SAP interface,which is a common connector framework (CCF) back end. These beans willuse the Transaction Service.

- Execution options
  - -family *familyName* -- Specifies the application familyname to be generated. By default, this name is set to the name of theEJB JAR file with the word Family appended. This option can bespecified more than once, as long as the values are unique.
  - -finderHelper *finderHelperClassNameremoteInterface*-- Specifies the finder helper class name(*finderHelperClassName*) and remote interface name(*remoteInterface*) for entity beans with CMP. If unspecified,it is assumed that no finder helper class is provided by the deployer.This option can be specified more than once, as long as the values areunique. For more information on finder helper classes, see Defining finder methods.
  - -usewstringindo -- Directs the tool to map the container-managedfields of an entity bean to the wstring IDL type (rather than the string type)on the DO. It is preferable to map to the string IDL type if the datasource contains single-byte character data; it is preferable to map tothe wstring IDL type if the data source contains double-byte or Unicodecharacter data.
  - -workloadmanaged -- Directs the tool to configure a CMP entity beanor a stateless session bean into a workload managing container and with aworkload managed home interface. For a BMP entity bean or a statefulsession bean it directs the tool to configure the bean only with a workloadmanaged home interface.
  - -clientdep *deployed-jarFile* -- Specifies the name of adependent JAR required by an EJB client that uses the enterprise bean beingdeployed. You must specify the full path of the file. To createmultiple client JAR files, you must specify this option for each JARfile. This option can be specified more than once, as long as thevalues are unique.
  - -serverdep *deployed-jarFile* -- Specifies the name of adependent JAR required by the EJB server (CB) that runs the deployedenterprise bean. You must specify the full path of the file. Tocreate multiple dependent JAR files, you must specify this option for each JARfile. This option can also be used to identify existing JAR files thatcontain classes required by the enterprise bean being deployed; when thisis done, the EJB server's CLASSPATH environment variable is automaticallyupdated to include this specified JAR file. This option can bespecified more than once, as long as the values are unique.
  - -sentinel *sentinelValue* -- Specifies an value for a Javatype or container-managed field for the deployed beans. If you set avalue for a Java type, do not put spaces around the = (equals) sign.
  - -strbehavior -- Specifies how the tool should determine the behaviorof the strings for a container-managed string fields in deployed beans.The corba value indicates that strings should be handled as CORBAstrings; the strip value directs the tool to remove trailing spaces fromstrings.

For session beans or entity beans with BMP, the code generation processruns without additional user intervention. For entity beans with CMP,the Object Builder GUI is displayed during execution of the command, and youmust create a DO implementation to manage the entity bean's persistentdata. For more information, see Building a data object during CMP entity bean deployment.

The **cbejb** tool deploys enterprise beans by generating extensiblemarkup language (XML) files and importing those files into ObjectBuilder. If the XML import fails, you can view any error messagesgenerated by Object Builder in the import_model.log file located in theproject directory.

If your CLASSPATH environment variable is too long, the **cbejb**command file fails. If this happens, shorten your CLASSPATH by removingany unnecessary files.The **cbejb** tool generates the following files for an EJB JAR filecontaining an enterprise bean named Account:

- AccountS.jar and (*Windows NT and Windows 2000*)AccountS.dll *or* (*AIX* or *Solaris*)libAccountS.so--The files required by the EJB server (CB) thatcontains this enterprise bean. The AccountS.jar file containsthe code generated from the Account EJB JAR file. TheAccountS.dll and libAccountS.so files contain the required C++classes.

  (*Windows NT and Windows 2000*) To run the Account enterprise beanin an EJB server (CB), the AccountS.jar file must be defined in theserver's CLASSPATH environment variable, and the AccountS.dll filemust be defined in the server's PATH environment variable.Typically, the System Management End User Interface (SM EUI) sets theseenvironment variables during installation of the deployed enterprise bean intoan EJB server (CB).

  (*AIX* or *Solaris*) To run the Account enterprise beanin an EJB server (CB), the AccountS.jar file must be defined in theserver's CLASSPATH environment variable, and the libAccountS.sofile must be defined in the server's LD_LIBRARY_PATH environmentvariable. Typically, the SM EUI sets these environment variables duringinstallation of the deployed enterprise bean into an EJB server (CB).
- AccountC.jar--The file required by an EJB client, includingenterprise beans that access other enterprise beans. This JAR filecontains everything in the original EJB JAR file except the enterprise beanimplementation class. To use the Account enterprise bean, a Java EJBclient must have the AccountC.jar and the IBM Java ORB defined in itsCLASSPATH environment variable.
- (*PAO only*) *paotoejbName*.jar--This fileis created by the **PAOToEJB** tool and is used to wrap an existingprocedural adapter object (PAO) in an enterprise bean.
- EJBAccountFamily.DDL--This file is used during installation ofthe Account family into an EJB server (CB) to update the database used by theSM EUI. Its name is composed of the EJB JAR file name with the stringFamily.DDL appended.

#### Building a data object during CMP entity bean deployment

When deploying an entity bean with CMP in the EJB server (CB), you mustcreate a DO implementation by using Component Broker's ObjectBuilder. This DO implementation manages the entity bean'spersistent data.To build a DO implementation, you must map the entity bean'scontainer-managed fields to the

appropriate data source as described in Guidelines for mapping the container-managed fields to a data source. Then, you must do one of the following:

- Use an existing DB2, Informix, or Oracle database to store the bean'spersistent data; for more information, see Using an existing DB2 or Oracle data source to store persistent data.
- Use an existing CICS or IMS application to store the bean'spersistent data; for more information, see Using an existing CICS or IMS application to store persistent data.
- Define a new DB2, Informix, or Oracle database to store the bean'spersistent data; for more information, see Defining a new DB2 or Oracle database to store persistent data.

Guidelines for mapping the container-managed fields to a data sourceWhen you deploy enterprise beans with the **cbejb** tool, a ComponentBroker DO IDL interface is created. The IDL attributes of thisinterface correspond to the entity bean's container-managedfields. You must then define the DO implementation by using ObjectBuilder to map the DO attributes to the attributes of a Persistent Object (PO)or Procedural Adapter Object (PAO), which correspond to the data types foundin the data source.

This section contains information on how the **cbejb** tool maps thecontainer-managed fields of entity beans to DO IDL attributes, and how theenterprise bean deployer maps DO IDL attributes to the entity bean's datasource. These guidelines apply whether you are using an existing datasource (also known as meet-in-the-middle deployment) or defining a new one(also known as top-down deployment).

- EJBObject or EJBHome variables--Objects of classes that implement theEJBObject or EJBHome interface map to the Object IDL type. At run time,this DO attribute contains the CORBA proxy for the EJBObject or EJBHomeobject. The CB EJB run time automatically converts between theEJBObject or EJBHome object (stored in the bean's container-managedfield) and the CORBA::Object attribute (stored in the C++DO). It is possible to deploy container-managed beans that havecontainer-managed fields of the same type, for example, a linked listimplementation where each node of the list is a container-managed bean thathas a reference to the next node. It is also possible to have circularreferences in a container-managed field, for example, a container-managed BeanA can have a container-managed field of type Bean B, which in turn has acontainer-managed field of type Bean A. When defining the DO-to-POmapping in Object Builder, you can use either a predefined Component Brokermapping of CORBA::Object to the data source, or implement a C++DO-to-PO mapping helper (in the standard Component Broker way) to invokemethods on the C++ proxy to obtain the persistent data. For moreinformation on creating a C++ DO-to-PO mapping, see the Component Broker Programming Guide.

  **Note:**

  Although Component Broker allows an entity bean's container-managedfields to be EJBObject or EJBHome objects, the Enterprise JavaBeans 1.0specification does not.
- Primary key variables--Do not map an enterprise bean's primarykey variables to the SQL type long varchar in a DB2, Informix or Oracledatabase. Instead, use either a varchar or a char type and set thelength appropriately.
- java.lang.String variables--Objects of this class aremapped to a DO IDL attribute of type string or wstring, depending on thecommand-line options used when the entity bean was deployed by using the**cbejb** tool (see Manually deploying an enterprise bean). By default, a variable of typejava.lang.String is mapped to a DO IDL attribute of typestring; however, the -usewstringindo option of the **cbejb** toolcan be used to map java.lang.String variables to DO IDLattributes of type wstring. (Mapping some of a bean's Stringfields to the IDL string type and others to the IDL wstring type is notsupported.) It is preferable to map to the string IDL type if the datasource contains single-byte character data; it is preferable to map tothe wstring IDL type if the data source contains double-byte or Unicodecharacter data.
- java.io.Serializable variables--Objects of classes thatimplement this interface are mapped to a DO IDL attribute of type ByteString(which is a typedef for sequence of octet defined in theIManagedClient.idl file). The EJB server (CB) automaticallyconverts serializable objects (stored in the entity bean'scontainer-managed fields) to the C++ sequence of octets containing theserialized form of the object (stored in the DO). Use the ComponentBroker default DO-to-PO mapping for ByteString to store the serialized objectdirectly in the data source.

  Unless you implement a C++ DO-to-PO mapping helper that passes the C++ByteString to a Java implementation by way of the interlanguage object model(IOM), it is not possible to manipulate the serialized Java object containedin a ByteString from within a C++ DO implementation. Therefore, if youare doing top-down enterprise bean development and you don't want tostore a serialized Java object in the data source, it is recommended that youavoid defining container-managed fields of type Serializable. Instead,make the Serializable variable a nonpersistent variable, define primitive typecontainer-managed fields to capture the state of the Serializable variable,and convert between the Serializable variable and the primitive variable inthe ejbLoad and ejbStore methods of the enterprise bean.
- Array variables--These variables are mapped to a DO IDL sequence ofthe corresponding type in the same way that the individual types are mapped toDO IDL attributes. For example, an array of thejava.lang.String class is mapped to a DO IDL attribute that is asequence of type string (or a sequence of type wstring, if the -usewstringindooption of the **cbejb** tool is used). The EJB server (CB)automatically converts between the array (stored in the entity bean'scontainer-managed fields) and the C++ sequence (stored in the DO). Youcan store the entire sequence in the data source as a whole, or you can writea C++ DO-to-PO mapping helper (in the standard Component Broker way) toiterate through the sequence and store individual elements in the data sourceseparately. For more information on creating a C++ DO-to-PO mapping,see the Component Broker Programming Guide.
- Date/Time fields--The **cbejb** tool maps container-managedfields of type java.util.Date and its subclasses(java.sql.Date, java.sql.Time,java.sql.Timestamp *only*) differently from otherSerializable fields. The following mapping rules are used:
  - java.util.Date: ISO-formatted timestamp string(yyyy-mm-dd-hh.mm.ss.mmmmmm)
  - java.sql.Date: ISO-formatted date string(yyyy-mm-dd)
  - java.sql.Time: ISO-formatted time string(hh.mm.ss)
  - java.sql.Timestamp: ISO-formatted timestamp string(yyyy-mm-dd-hh.mm.ss.mmmmmm)

  Therefore a container-managed field of one of the above types should bemapped to either a string or a database-specific date/time field that can takean ISO-formatted string as input. (For example, both DB2 and OracleDate/Time/Timestamp column types can take ISO strings as input values.)If a deployer chooses to map a Date/Time container-managed field to somethingother than the types mentioned above, then a special data mapping code shouldbe written in the DO implementation. The mapping code must be able toconvert an ISO-formatted string to a backend-specific type and viceversa.

  The java.sql.Timestamp class has a precision of nanoseconds,whereas ISO timestamp format has a precision of microseconds.Therefore, precision

is compromised (by rounding nanoseconds to nearestmicroseconds) when a Timestamp CMP field is mapped. Users should beparticularly aware of this when they use the java.sql.Timestampclass as one of the attributes of bean's primary key.

While mapping java.sql.Date to ISO Date format, the timefield values are ignored. Similarly while mappingjava.sql.Time to ISO Time, the date field values areignored.

**Note:**

For DB2 only: If an existing database outputs date/time in a non-ISOformat, then the deployer must rebind DB2 packages using the "DATETIME ISO"option.

Using an existing DB2 or Oracle data source to store persistent dataTo use an existing DB2 or Oracle database to store a CMP entity bean'spersistent data, follow these steps. The end result is a PO withattributes that correspond to the items in the database schema.

1. When Object Builder starts, it presents the Open Project dialog.Choose the location of the project directory for your enterprise bean andclick **Finish**.
2. To import an existing relational database schema, click **DBA-DefinedSchemas** and right-click the appropriate database type.
    a. On the pop-up menu, click **Import** and **SQL**.
    b. On the **Import SQL** dialog box, click **Find** and browsefor your SQL file.
    c. Double-click your SQL file.
    d. Change the name in the **Database Name** text field from`Database` to the actual name of the database.
    e. Select the appropriate database type and click **Finish**.
3. To create a persistent object (PO) from the database schema, expand**DBA-Defined Schemas** and expand your group.
    a. Highlight your schema and then right-click it to display a pop-upmenu. Click **Add->Persistent Object**.
    b. On the **Names and Attributes** dialog box, accept the defaults andclick **Finish**.
4. Create a DO implementation as follows:
    a. Expand the **User-Defined DOs**, expand the **DO File** (forexample CBAccountDO), expand the **DO Interface** (for example,com_ibm_ejs_doc_account_AccountDO), and select the **DOImplementation**.
    b. On the **DO Implementation** pop-up menu, select**Properties**.
    c. On the **Name and Platform** page, select the **DeploymentPlatform** (for example, NT, AIX, or Solaris) and click**Next**.
    d. On the **Behavior** page, make the appropriate selections and click**Next**:
        - *For DB2*, select `BOIM with any Key` for**Environment**, select `Embedded SQL` for **Form ofPersistent Behavior and Implementation**, select `Delegating` for**Data Access Pattern**, and select `Home name` and`key` for **Handle for Storing Pointers**.
        - *For Oracle*, select `BOIM with any Key` for**Environment**, select `Oracle Caching services` for**Form of Persistent Behavior and Implementation**, select`Delegating` for **Data Access Pattern**, and select `Homename` and `key` for **Handle for StoringPointers**.
    e. On the **Implementation Inheritance** page, make the appropriateselections for the parent class and click **Next**:
        - *For DB2*, select`IRDBIMExtLocalToServer::IDataObject`
        - *For Oracle*, select`IRDBIMExtLocalToServer::ICachingServiceDataObject`
    f. Accept the defaults for the **Attributes**, **Methods**, and**Key and Copy Helper** pages by clicking **Next** on eachpage.
    g. On the **Associated Persistent Objects** page, click **AddAnother**. Accept the default for the instance name (iPO) andselect the correct type. Click **Next**.
    h. On the **Attribute Mapping** page, map the container-managed fieldsof the entity bean to the corresponding items in the database schema.Object Builder creates default mappings for the data object attributes forwhich it can identify corresponding persistent object attributes. Thedefault mapping is generally suitable for everything except for the primarykey variable, which you must map to a varchar or char type rather than a longvarchar type. For more information, see Guidelines for mapping the container-managed fields to a data source. After you finish mapping the attributes, click**Finish**.
    i. *Oracle only*. When mapping an entity bean with CMP to anOracle database, expand the **Container Definition** folder andright-click the EJB container. From the pop-up menu, click**Properties**. In the wizard, click **Next** until youreach the **Data Access Patterns**; on that page, check the**Cache Service** checkbox and click **Finish**.
    j. Exit from Object Builder by clicking **File->Exit**; save anychanges if prompted.
    k. Create the database specified by the **Database** text field anduse the SQL file specified by the **Schema File** text field to createa database table. For more information on creating a database anddatabase table with an SQL file, consult your DB2 or Oracledocumentation. The SQL file can be found in the following directory,where *projDir* is the project directory created by the**cbejb** tool:
        - *On Windows NT and Windows 2000, projDir*\Working\NT
        - *On AIX, projDir*/Working/AIX
        - *On Solaris, projDir*/Working/Solaris

Using an existing CICS or IMS application to store persistent dataTo use CICS or IMS for Persistent Adaptor Object (PAO) storage, followingthese instructions. Note that if the persistent store uses a CICS orIMS application (by way of a PAO), only application data is used; themethods on the CICS or IMS application are pushdown methods, which runapplication-specific logic rather than storing and loading data.

The following prerequisites must be met to map an entity bean with CMP toan existing CICS or IMS application:

- The entity bean's transaction attribute must be set to TX_MANDATORYif you want to map the bean to a HOD- or ECI-based application. Thetransaction attribute must be set to either the TX_MANDATORY or TX_REQUIRED ifyou want to map it to an APPC-based application.
- The existing CICS or IMS application must be represented as a proceduraladapter object (PAO). See the Procedural Application Adaptor

Development Guide for more informationon creating PAOs.

- The PAO class files must be specified in the CLASSPATH environmentvariable.
- The entity bean must implement all enterprise bean logic; the onlyremaining requirement is to map the entity bean's container-managedfields to the PAO. Pushdown methods on the PAO cannot be utilized fromthe enterprise bean. (PAO pushdown methods can be used from an entitybean with CMP generated by using the **PAOToEJB** tool as described in Creating an enterprise bean from an existing CICS or IMS application.)
- The **cbejb** tool must be run as follows, where the*ejb-jarFile* is the EJB JAR file containing the entity bean:

  ```
  # cbejb ejb-jarFile [-hod | -eci | -appc[beanNames]]
  ```

  For a description of the **cbejb** tool's syntax, see Manually deploying an enterprise bean.

If you have met the prerequisites, use Object Builder to create the mappingbetween the entity bean and the CICS or IMS application:

1. When Object Builder starts, it presents the Open Project dialog.Choose the location of the project directory for your enterprise bean andclick **Finish**.
2. From the main menu, click **Platform** and then**Target**. Uncheck the 390 platform.
3. Click **User-Defined PA Schemas** and right-click theselection.
4. From the pop-up menu, click **Import** and then**Bean**. On the **Import Bean** dialog box, type theclass name of the PAO bean and click **Next**.
5. Select the appropriate connector type and click **Next**.
6. Select the primary key attribute name from the **Properties**list.
7. Click >> to move the primary key to the **KeyAttributes** list and click **Finish**.
8. *For HOD and ECI only*, do the following for both the MO and theHomeMO:
   a. In the **Tasks and Object** panel, expand the **User-DefinedBusiness Objects**, expand the object, and expand the object'sBO. From the MO file's pop-up menu, click**Properties**.
   b. Change the **Service to use** property from `TransactionService` to `Session Service`.
9. Create a DO implementation as follows:
   a. On the **Tasks and Object** panel, expand the **User-DefinedDOs**, expand the **DO File** from the menu, and click the **DOInterface**.
   b. On the **DO Interface** pop-up menu, select **AddImplementation**.
   c. On the **Behavior** page, select `BOIM with any Key` for**Environment**, select `Procedural Adapters` for **Form ofPersistent Behavior and Implementation**, select `Delegating` for**Data Access Patterns**, and select `Default` for **Handlefor Storing Pointers**. Click **Next**.
   d. Click **Next** on the **Implementation Inheritance** page,the **Attributes** page, the **Methods** page, and the **Keyand Copy Helper** page.
   e. On the **Associated Persistent Object** page, click **AddAnother**, verify that the PO that you previously created is selected, andclick **Next**.
   f. On the **Attribute Mapping** page, designate how thecontainer-managed fields of the entity bean correspond to the items in theexisting PAO. This designation is done by defining a mapping betweenthe attributes of the DO (which match the entity bean's container-managedfields) to the attributes of the PO (which match the existing PAO). Inthe **Attributes** list, there is a DO attribute corresponding to eachof the bean's container-managed fields.

      For each DO attribute in the **Attributes** list, right-click theattribute and click **Primitive** from the menu. From the**Persistent Object Attribute** drop-down menu, select the PO attribute(the item from the existing database schema) that corresponds to the DOattribute. For more information, see Guidelines for mapping the container-managed fields to a data source. After you have processed all container-managedfields, click **Next**.
   g. On the **Methods Mapping** page, for each method in the list of**Special Framework Methods**, right-click a method and click **AddMapping**. From the **Persistent Object Method** drop-downmenu, select the PO method with the same name as the selected DOmethod. If there are more methods than available mappings, map methodsto similarly named methods. For example, map update to update().After you have processed all of the methods, click **Finish**.
   h. Expand the **Container Definition** folder and right-click the EJBcontainer. From the pop-up menu, click **Properties**. Inthe wizard, click **Next** until you reach the **Data AccessPatterns** page.
   i. On the **Data Access Patterns** page, select one of the followingitems and then click **Next**:
      - *For HOD or ECI*, select `Use PAA Sessionservices`.
      - *For APPC*, select `Use PAA Transactionservices`.
   j. On the **Service Details** page, do the following and then click**Next**:
      - *For HOD or ECI*, select `Throw an exception and abandon thecall` for **Behavior for Methods Called Outside aTransaction**; define a connection name, for example,`MY_PAA_Connection`; select `Host on Demand` or`ECI connection`, respectively, for the **Type ofconnection**.
      - *For APPC*, select `Throw an exception and abandon thecall` for enterprise beans with the TX_MANDATORY transaction attribute,or select `Start a new transaction and complete the call` forenterprise beans with the TX_REQUIRED transaction attribute.
   k. Select `Caching` for **Business Object**.
   l. Select `Delegating` for **Data Object**.
   m. Click **Finish**.
10. Exit from Object Builder by clicking **File->Exit**; save anychanges if prompted.

Defining a new DB2 or Oracle database to store persistent dataWhen you use a top-down development approach to enterprise bean development,enterprise bean deployment must occur in three phases:

1. Define the database schema, map the container-managedfields of the entity bean with CMP to the database schema, and generate thecode to encapsulate this mapping. For more information, see Mapping the database schema.
2. Create the database and database tables. For more information, see Creating the database and database table.
3. Compile the code generated in phase 1; compilation fails if the database and database tablesdo not exist.

#### Mapping the database schema

After you have defined the manner in which the entity bean maps to adatabase, create the mapping by running the **cbejb** tool with the -ncoption to prevent automatic compilation after code generation. Forexample, to create a mapping for an Account bean stored in an EJB JAR filenamed EJBAccount.jar, enter the following command:

```
# cbejb EJBAccount.jar -nc -queryable [-oracle |-cacheddb2]
```

**Note:**

If the database being used to store the persistent data is either Oracle orDB2, those options must also be specified.

#### Creating the database and database table

Follow these instructions to create a database and database table by usingthe Object Builder GUI:

1. When Object Builder starts, it presents the Open Project dialog.Choose the location of the project directory for your enterprise bean andclick **Finish**.
2. Create a DO implementation as follows:
    a. Expand the **User-Defined DOs**, expand the **DO File** fromthe menu, and click the **DO Interface**.
    b. On the **DO Interface** pop-up menu, select **AddImplementation**. If the implementation is already present, you canmodify it by selecting the implementation, invoking the pop-up menu, andselecting **Properties**.
    c. On the **Name and Platform** page, select the platform and click**Next**.
    d. On the **Behavior** page, make the appropriate selections and click**Next**:
        - *For DB2*: select `BOIM with any Key` for**Environment**, select `Embedded SQL` for **Form ofPersistent Behavior and Implementation**, select `Delegating` for**Data Access Pattern**, and select `Home name` and`key` for **Handle for Storing Pointers**.
        - *For Oracle*: select `BOIM with any Key` for**Environment**, select `Oracle Caching services` for**Form of Persistent Behavior and Implementation**, select`Delegating` for **Data Access Pattern**, and select `Homename` and `key` for **Handle for StoringPointers**.
    e. On the **Implementation Inheritance** page, make the appropriateselections for the parent class and click **Next**:
        - *For DB2*, select`IRDBIMExtLocalToServer::IDataObject`
        - *For Oracle*, select`RDBIMExtLocalToServer::ICachingServiceDataObject`
        - *For CICS or IMS PAO*, select`IRDBIMExtLocalToServer::IDataObject`
    f. Accept the defaults for the **Attributes**, **Methods**, and**Key and Copy Helper** pages by clicking **Next** on eachpage.
    g. On the **Associated Persistent Objects** page, click **AddAnother**. Accept the default for the instance name (iPO) andselect the correct type. Click **Next**.
    h. On the **Attribute Mapping** page, map the container-managed fieldsof the entity bean to the corresponding items in the database schema.The default mapping is generally suitable for everything except for theprimary key variable, which you must map to a varchar or char type rather thana long varchar type. Object Builder creates default mappings for thedata object attributes for which it can identify corresponding persistentobject attributes. For more information, see Guidelines for mapping the container-managed fields to a data source. After you finish mapping the attributes, click**Finish**.
    i. *Oracle only*. When mapping an entity bean with CMP to anOracle database, expand the **Container Definition** folder andright-click the EJB container. From the pop-up menu, click**Properties**. In the wizard, click **Next** until youreach the **Data Access Patterns**; on that page, check the**Cache Service** checkbox and click **Finish**.
    j. Exit from Object Builder by clicking **File->Exit**; save anychanges if prompted.
    k. Create the database specified by the **Database** text field anduse the SQL file specified by the **Schema File** text field to createa database table. For more information on creating a database anddatabase table with an SQL file, consult your DB2 or Oracledocumentation. The SQL file can be found in the following directory,where *projDir* is the project directory created by the**cbejb** tool:
        - *On Windows NT*, *projDir*\Working\NT
        - *On AIX*, *projDir*/Working/AIX
        - *On Solaris*, *projDir*/Working/Solaris

#### Compiling the generated code

After both the database and database table are created, compile theenterprise bean code by using the following commands:

- **On Windows NT.**

  ```
  # cd projDir\Working\NT

  # nmake -f all.mak
  ```

- **On AIX.**

  ```
  # cd projDir/Working/AIX
  ```

```
# make -f all.mak
```

- **On Solaris.**

```
# cd projDir/Working/Solaris

# make -f all.mak
```

#### Installing an enterprise bean and configuring its EJB server (CB)

Follow these steps to install an enterprise bean and configure the resultingEJB server (CB):

1. (*Entity bean with CMP using DB2 only*) Use the bind file, whichObject Builder generates as a side effect of using the **cbejb** tool,to bind the enterprise bean to the database (for example, `db2 bindAccountTblPO.bnd`).
2. Using the SM EUI, install the application generated by**cbejb**. In general, this installation is the same asinstalling a Component Broker application generated by Object Builder:
    a. Load the application into a host image.
    b. Add the application to a configuration.
    c. Associate the EJB application with a server group or server. (Ifthe server group or server does not already exist, you must create it.)
    d. (*Entity bean with CMP only*) Associate the entity bean'sdata source (DB2, Oracle, CICS, or IMS PAA) with the EJB application:
        - DB2: associate the DB2 services (iDB2IMServices) with the EJBserver.
        - Oracle: associate the Oracle services (iOAAServices) with the EJBserver.
        - CICS or IMS PAA: associate the PAA services (iPAAServices) with theEJB server.
    e. Configure the EJB server (CB) with a host.
    f. Set the ORB request timeout for both clients and servers to 300seconds.
    g. If the EJB server requires Java Virtual Machine (JVM) properties to beset, edit the JVM properties. Do this in the server model instead ofthe server image. For instance, if the enterprise bean performs a JNDIlookup to access other enterprise beans, the server hosting the enterprisebean must have its JVM properties set to include values for JNDIproperties.
    h. Activate the EJB server configuration.
    i. Start the EJB server.

#### Binding the JNDI name of an enterprise bean into the JNDI namespace

**Note:**

This section does not apply to servers running on the AIX, Windows NT,Windows 2000, or Solaris platforms.

An enterprise bean's JNDI home name is defined within its deploymentdescriptor as described in The deployment descriptor. This name is used by EJB clients (including otherenterprise beans) to find the home interface of an enterprise bean.

The **ejbbind** tool locates the CB home that implements theenterprise bean's EJBHome interface in the Component Brokernamespace. It also rebinds the home name into the namespace, using theJNDI home name specified in the enterprise bean's deploymentdescriptor. This binding enables an EJB client to look up the EJB homeby using the JNDI name specified in the bean's deploymentdescriptor. An enterprise bean can be bound on a different machine fromthe one on which the bean was deployed.

The subtree of the Component Broker namespace in which the JNDI name isbound can be controlled by the command-line options used with the**ejbbind** tool. The manner in which the name is bound (thesubtree chosen) affects the JNDI name that EJB clients must use to look up theenterprise bean's EJB home and also affects the visibility of theenterprise bean's EJB home. Specifically, the JNDI name can bebound in one of the following ways:

- The JNDI name can be bound into the local root. Under this bindingapproach, EJB clients use the JNDI name in the enterprise bean'sdeployment descriptor. The approach restricts the visibility of the EJBhome to EJB clients using the same name server (the same bootstrap host) andcan cause collisions with other names in the tree.
- The JNDI name can be bound into the host name tree (athost/resources/factories/EJBHomes). Under this binding approach, EJBclients must prefix the string `host/resources/factories/EJBHomes` tothe JNDI name given in the bean's deployment descriptor. Thisapproach minimizes collisions with other names in the tree, but restrictsvisibility of the enterprise bean home to clients using the same nameserver.
- The JNDI name can be bound into the workgroup name tree (atworkgroup/resources/factories/EJBHomes). Under this binding approach,EJB clients must prefix the string`workgroup/resources/factories/EJBHomes` to the JNDI name given inthe enterprise bean's deployment descriptor, and the EJB home is visibleto all EJB clients using a name server that belongs to the same preferredworkgroup.
- The JNDI name can be bound into the cell name tree (atcell/resources/factories/EJBHomes). Under this binding approach, EJBclients must prefix `cell/resources/factories/EJBHomes` to the JNDIname in the bean's deployment descriptor, and the EJB home is visiblethroughout the cell.

Before running the **ejbbind** tool, do the following:

- Deploy your enterprise bean for Component Broker by using the**cbejb** tool. For more information, see Manually deploying an enterprise bean.
- Install the Component Broker application that **cbejb** toolgenerates, and configure it on a specific EJB server (CB) by using the SMEUI. For more information, see Installing an enterprise bean and configuring its EJB server (CB).
- Start the CBConnector Service and a name server, if they are not alreadyrunning. For more information, see the Component Broker System Administration Guide.
- Activate the configuration containing the EJB server (CB) that runs theapplication.
- Determine the IP address (the bootstrap host name) and port number (thebootstrap port) of the machine running the name server.

Invoke the **ejbbind** command with the following syntax:

```
ejbbind ejb-jarFile [beanParm] [-f] [-BindLocalRoot ] [-BindHost]  [-BindWorkgroup] [-BindCell]
[-BindAllTrees] [-ORBInitialHost hostName] [-ORBInitialPort portNumber] [-u] [-UnbindLocalRoot]
[-UnbindHost] [-UnbindWorkgroup] [-UnbindCell] [-UnbindAllTrees]
```

The *ejb-jarFile* is the fully-qualified path name of the EJB JARfile containing the enterprise bean to be bound or unbound. Theoptional beanParm argument is used to bind a single enterprise beanin the EJB JAR file; you can identify this bean by supplying a fullyqualified name (for example,com.ibm.ejs.doc.account.Account, whereAccount is the bean name) or the name of the enterprise bean's deploymentdescriptor file without the .ser extension. If an enterprisebean has multiple deployment descriptors in the same EJB JAR file, you mustsupply the deployment descriptor file name rather than the enterprise beanname.

When no options are specified, the JNDI name is bound into the localroot's name tree, using the local host and port 900 for the bootstraphost (the name server).

The other options do the following:

- -f -- Force the bind, even if the JNDI name is already bound in thenamespace; this option is not valid with the unbind commandoptions.
- -BindLocalRoot -- Bind the JNDI name into the local root's nametree.
- -BindHost -- Bind the JNDI name into the host name tree.
- -BindWorkgroup -- Bind the JNDI name into the workgroup nametree.
- -BindCell -- Bind the JNDI name into the cell name tree.
- -BindAllTrees -- Bind the JNDI name into the host, the workgroup, andthe cell name trees.
- -ORBInitialHost hostName -- Identify the bootstrap host (thedefault is the local host).
- -ORBInitiallPort portNumber -- Identify the bootstrap port(the default is port 900).
- -u -- Unbind the JNDI name; this option is not valid with bindcommand options.
- -UnbindLocalRoot -- Unbind the JNDI name from the local root'sname tree.
- -UnbindHost -- Unbind the JNDI name from the host name tree.
- -UnbindWorkgroup -- Unbind the JNDI name from the workgroup nametree.
- -UnbindCell -- Unbind the JNDI name from the cell name tree.
- -UnbindAllTrees -- Unbind the JNDI name from the host, the workgroup,and the cell name trees.

If the command is successful, it issues a message similar to thefollowing:

```
Name AccountHome was bound to CB Home
```

You must run the **ejbbind** tool again if any of the followingoccurs:

- You modify the JNDI name of an enterprise bean. You can modify theJNDI name by using the **jetace** tool. For more information,see Creating an EJB module.
- You reconfigure Component Broker. In this case, you must rebindevery enterprise bean served by this configuration.
- You move the enterprise bean to a different EJB server (CB) or a differentmachine.

### Configuring systems management to enable lazy enumeration

To enable lazy enumeration (see Creating finder logic in the EJB server (CB)), follow these steps:

1. From the System Management End User Interface (SM EUI), go to the Viewmenu, and set the View Level to Control.
2. Expand **Host Images**
3. Expand the name of your host.
4. Expand **Server Images**.
5. Expand the name of your server.
6. Expand **Container Images**.
7. Right-click **iIteratorSysObjsNoPRef**. From the pop-upmenu, select **Properties**. Change the followingproperties:
    - Change the **Default transaction policy** to`throwException`.
    - Change the **Memory management policy** to `passivate at end oftransaction`.

The transaction policy ensures that the caller starts a transaction.The memory management policy ensures that the lazy enumerations are passivatedwhen the transaction completes.

---

## Resolving to EJB homes using lifecycle services in CBConnector

**Note:**

This section applies only to servers running on the AIX, Windows NT, Windows2000, or Solaris platforms.

When an EJB client performs a simple JNDI lookup, a 1-to-1 mapping is madebetween the name and the particular EJB home instance. In a distributedenvironment, this model can be limiting. In such an environment, forexample, there may be many EJB homes supporting the same type of enterprisebean. It is better to have an approach that does not require anapplication to request a specific instance of that home. In addition,as changes are

made to the system, it is important that applications not haveto be changed or redeployed to specify a different instance of an EJBhome. The CBConnector LifeCycle Service provides a level of indirectionand abstraction that allows the application to request a home that is within aparticular scope of location within the distributed environment, yet beisolated from the specifics of the exact configuration of theenvironment. For more info on lifecycle factory finders, see theLifeCycle section in the Advanced Programming Guide.

Using CBConnector, a JNDI context can be associated with a LifeCycleService factory finder so that the associated factory finder is used toresolve EJB home lookup operations from the context. Contexts such asthese enable deployers of EJB applications to take advantage of the power offactory finders in a manner that is transparent to clients of theseapplications.

To resolve EJB home lookups with factory finders, the application deployercan use pre-defined default application contexts associated with the variousCBConnector-supplied default factory finders or use the **appbind**tool to create application-specific contexts and associate them with any givenfactory finder. For more information on each approach, see Default context-to-finder associations and Application-specific contexts and the appbind tool.

**Note:**

Default application contexts and application-specific contexts eliminate theneed for the **ejbbind** tool, which creates a simple 1-to-1 mapping ofa JNDI name and an EJB home instance. Clients must use one of thedefault initial context factories or an application-specific context factorygenerated by the **appbind** tool.

## Default context-to-finder associations

There are several default factory finders built into CBConnector, each ofwhich searches particular scopes of location when finding a factory.When an EJB application is deployed on a CBConnector server, the EJB homes forthe application are bound in the LifeCycle repository using the names for theEJB homes as specified by the deployment descriptors contained in theapplication's EJB jar file. A factory finder can find any EJB homewithin the scope of its particular search rules.

An EJB client can use a particular built-in CBConnector default factoryfinder simply by using the initial context factory that corresponds to thatfactory finder. The initial context returned by the context factorywill use its corresponding factory finder to resolve EJB home lookuprequests.

Contexts returned by the following initial context factories:

1. com.ibm.ejb.cb.runtime.CBCtxFactoryHostDefault
2. com.ibm.ejb.cb.runtime.CBCtxFactoryHostWidenedDefault
3. com.ibm.ejb.cb.runtime.CBCtxFactoryHostServerDefault
4. com.ibm.ejb.cb.runtime.CBCtxFactoryHostServerWidenedDefault
5. com.ibm.ejb.cb.runtime.CBCtxFactoryWorkGroupDefault
6. com.ibm.ejb.cb.runtime.CBCtxFactoryWorkGroupWidenedDefault
7. com.ibm.ejb.cb.runtime.CBCtxFactoryWorkGroupServerDefault
8. com.ibm.ejb.cb.runtime.CBCtxFactoryWorkGroupServerWidenedDefault
9. com.ibm.ejb.cb.runtime.CBCtxFactoryCellDefault
10. com.ibm.ejb.cb.runtime.CBCtxFactoryCellServerDefault
11. com.ibm.ejb.cb.runtime.CBCtxFactoryCellServerWidenedDefault

resolve EJB home lookup operations with the corresponding factoryfinders:

1. host/resources/factory-finders/host-scope
2. host/resources/factory-finders/host-scope-widened
3. host/resources/factory-finders/*server*-server-scope
4. host/resources/factory-finders/*server*-server-scope-widened
5. workgroup/resources/factory-finders/workgroup-scope
6. workgroup/resources/factory-finders/workgroup-scope-widened
7. workgroup/resources/factory-finders/*server*-server-scope
8. workgroup/resources/factory-finders/*server*-server-scope-widened
9. cell/resources/factory-finders/host-scope
10. cell/resources/factory-finders/*server*-server-scope
11. cell/resources/factory-finders/*server*-server-scope-widened

Server-based context factories can only be used by a client that is runningas a CBConnector server, in which case, *server* is the name of theCBConnector server.

Default context factories can only be used by client applications thatissue fully qualified EJB home lookups. If a client traverses to asubcontext and then performs a partially qualified EJB home lookup, you mustrun the **appbind** tool to create an application-specific context withhome subcontexts and to generate an application-specific initial contextfactory. For more information, see Application-specific contexts and the appbind tool.

## Application-specific contexts and the appbind tool

If a CBConnector-supplied default factory finder is being used to locate anEJB home, CBConnector supplies a default mapping between application contextsand default factory finders (for more information, see Default context-to-finder associations). For added flexibility, an enterprise bean deployercan create an application-specific context with optional EJB home subcontextsand associate it with any factory finder. The factory finderassociation can be

changed at a later time if desired. To isolateclients from the actual context name, the enterprise bean deployer generatesan initial context factory for the application-specific context by using the**appbind** tool.

The **appbind** tool allows deployers to create anapplication-specific naming context and associate it with a selected factoryfinder so that lookup operations are resolved with that factory finder.These application-specific contexts are designed to be initial JNDI contextsfor EJB clients so that JNDI lookup calls on EJB homes are transparentlyresolved with the associated factory finder. The **appbind**tool enables users to create, modify, and delete such application-specificcontexts. Note that the application's EJB home instances are notactually bound under the application-specific context. Instead, theyare bound to the LifeCycle repository. The associated factory finderwill resolve the EJB home lookups using the lifecycle rules defined forit.

Using the **appbind** tool also helps to avoid naming collisions forenterprise beans that are written to version 1.1 of the EJBspecification. It can be used to create separate JNDI namespaces forenterprise beans that have the same JNDI name but are deployed with initialcontext factories located at different places in the namespace. Thisprevents naming conflicts between these beans.

All application-specific contexts must have one of the following contextname stems:

- host/applications/initial-contexts
- workgroup/applications/initial-contexts
- cell/applications/initial-contexts

depending on whether a scope of host, workgroup, or cell is specified whenthe context is created.

By default, the factory finderhost/resources/factory-finders/host-scope-widened is associated with anapplication-specific context created with the **appbind** tool.However, the user can specify another factory finder. The factoryfinder can be one of the other default factory finders, one created by anadministrator using System Management, or one created by an applicationprogram you write. For more information, see the LifeCycle section inthe Advanced Programming Guide.

Under an application-specific context, subcontexts for EJB home namesoptionally can be created. For example, if the name for a home iscom/mycom/myapp/MyHome, the subcontext com/mycom/myapp can be created.These subcontexts provide additional transparency to the client. Theyallow a client to traverse the JNDI name space from the application-specificcontext down to any subcontext that corresponds to a non-leaf component of anEJB home name. The factory finder associated with theapplication-specific context is also used to resolve EJB home lookupoperations from these subcontexts. The **appbind** tool createsa subcontext for each home name in the deployment descriptors within aspecified EJB JAR file.

The **appbind** tool can optionally create a Java source file for aninitial context factory for the application-specific context beingcreated. This initial context factory can be used as the initialcontext factory by clients. The **appbind** tool also allows theuser to override the default bootstrap host to use for ORBinitialization.Invoke the **appbind** tool with the following syntax:

```
appbind [-u] -name  contextName [-sc jarFileName] [-host | -workgroup | -cell][-factoryfinder
factoryFinderPath][-genctxfactory  factoryClassName [-o  targetDir]][-boothost  bootstrapHostUrl]
```

The context being bound or unbound is specified with the required -nameoption, where contextName is the name of the JNDIapplication-specific context to bind or unbind. All application contextnames are relative to one of the following context name stems

- host/applications/initial-contexts
- workgroup/applications/initial-contexts
- cell/applications/initial-contexts

depending on whether a scope of host, workgroup, or cell wasspecified. (See the -host, -workgroup, and -cell options below.)

A bind operation is performed unless the -u option is specified, in whichcase, an unbind operation is performed. If a bind operation isperformed on an existing context, the current factory finder association isadded or replaced. The context cannot be a child or parent of a contextwhich already has a factory finder association.

The other options do the following:

- -u--This flag is used to perform an unbind operation. Anunbind operation unbinds the context specified with the -name option and the-sc option, if specified. If the -sc option is specified, only thesubcontexts corresponding to the JNDI home names in the JAR's deploymentdescriptors are removed. If the -sc option is not used, the contextspecified by the -name option and all of its subcontexts are unbound.To help keep the name tree manageable, once a context or subcontext isunbound, parent contexts are recursively unbound up to the context name stem(see the -name option above) or until a non-empty parent isencountered.
- -sc--This option is used to specify subcontexts, where filejarFileName is the name of an EJB JAR file that contains deploymentdescriptors with EJB home names. Each of the EJB home names, notincluding the leaf-name component, is treated as a subcontext name. Forexample, if the name for a home is com/mycom/myapp/MyHome, the subcontext nameis com/mycom/myapp.

  When binding, the subcontext names are created under theapplication-specific context specified by the -name flag. Whenunbinding, the contexts which are unbound are restricted to the subcontextnames identified by the JAR file. Whether binding or unbinding, othersubcontexts are not affected.
- -host, -workgroup, -cell--These flags control the scope of theapplication context being bound or unbound. Each scope has acorresponding context name stem, as described in the -name flag sectionabove. The -host, -workgroup, and -cell flags specify a scope of host,workgroup, or cell, respectively, for the context. The default scope ishost scope. Only one scope can be specified per bind or unbindoperation.
- -factoryfinder--This option is used to specify which factory finderto associate with the application-specific context being bound, wherefactoryFinderPath is the name of the factory finder. Thedefault factory finder ishost/resources/factory-finders/host-scope-widened.

  This option does not apply to unbind operations.
- -genctxfactory--Typically, when an application-specific context isbound, it is desirable to have an initial context factory for theapplication-specific context. This option directs the**appbind** tool to create a Java source file for an initial contextfactory, where factoryClassName is the fully-qualified class name ofthe context factory. All package prefix subdirectories are created, ifnecessary. If the source file already exists, it is replaced.The file and its containing subdirectories are created relative to thedirectory specified with the -o option or, by default, relative to the currentdirectory.

This option does not apply to unbind operations.

- -o--This option is used to specify the target directory for theinitial context factory file (see the -genctxfactory option), wheretargetDir is the directory path (not including package prefixdirectories). The default target directory is the currentdirectory.

  This option does not apply to unbind operations.

  If the -o option is used, use of the -genctxfactory flag isrequired.

- -boothost--This option is used to override the default host and portused for ORB initialization, where *bootstrapHostUrl* is the URL ofthe bootstrap host. The bootstrap host URL has the form

  ```
  iiop:// hostName [: portNumber]
  ```

---

## Creating an enterprise bean from an existing CICS or IMS application

You can create an enterprise bean from an existing CICS or IMS application byusing the **PAOToEJB** tool. The application must be mapped intoa PAO prior to creating the enterprise bean. For more information oncreating PAOs, see the Component Broker document entitled Procedural Application Adaptor Development Guide and the VisualAge forJava, Enterprise Edition documentation.

The **PAOToEJB** tool runs independently of the other toolsdescribed in this chapter. To create an enterprise bean from a PAOclass, do the following:

1. Change to the directory where your PAO class file exists.
2. Add the PAO class file's directory, or the JAR file containing theclass, to your CLASSPATH environment variable.
3. Invoke the **PAOToEJB** command with the following syntax:

   ```
   PAOToEJB -name [ejbName] paoClass -hod |  -eci | -appc
   ```

   The *ejbName* argument is optional and specifies the enterprisebean's name (for example, Account). If this name is not supplied,the enterprise bean is named by using the short name of the PAO class.The *paoClass* argument is required and specifies the fully qualifiedJava name of the PAO class without the .class extension; the PAOclass is always a subclass ofcom.ibm.ivj.eab.paa.EntityProceduralAdapterObject.You must also specify one of the following options:

   - -hod --This indicates that the PAO class is for Host On-Demand(HOD). HOD is a browser-based 3270 telnet connection.
   - -eci --This indicates that the PAO class is for External CallInterface (ECI). ECI is a proprietary protocol that provides a remoteprocedure call (RPC)-like interface into CICS.
   - -appc --This indicates that the PAO class is for advancedprogram-to-program communications (APPC), which is the System NetworkArchitecture (SNA) for LU 6.2 communications.

   **Note:**

   EJB clients that access entity beans with CMP that use HOD or ECI for CICS orIMS applications must begin a transaction before invoking a method on theseentity beans. This is necessary because these types of entity beansmust use the TX_MANDATORY transaction attribute.
4. If the *paoClass* is part of a Java package, then you must createthe corresponding directory structure and move the generated Java files intothis directory.
5. Compile the Java source files of the newly created enterprise bean:

   ```
   javac ejbName*.java
   ```
6. Place the compiled class components of the enterprise bean into a JAR orZIP file and use the **jetace** tool to create an EJB JAR file for thebean, as described in Creating an EJB module.
7. Deploy the EJB JAR file by using the **cbejb** tool as described inManually deploying an enterprise bean.

---

## Creating an enterprise bean that communicates with MQSeries

Component Broker contains tools for developing BOs that send or receiveMQSeries messages. It also allows access to MQSeries queues withindistributed transactions. The EJB server (CB) builds on this MQSeriessupport and allows you to create an enterprise bean that wraps anMQSeries-based BO.

The MQSeries EJB support enables an EJB client application to indirectlyinteract with MQSeries through an EJB client interface. Both theComponent Broker support for MQSeries BOs and the EJB support described hererequire you to modify the DO implementation generated by ObjectBuilder. The main difference between these two supported approaches isthat when Component Broker MQSeries-based BOs are built, the MQSeries messagecontent is specified through Object Builder, whereas the EJB support requiresthe MQSeries message content to be specified in a Java properties file.

For more information on the MQSeries support in Component Broker, see theMQSeries Application Adaptor Development Guide document.

The **mqaaejb** tool generates a session bean that wraps a ComponentBroker BO based on the MQSeries Application Adaptor. The resultingsession bean implementation is specific to the EJB server (CB) and is notportable to other EJB servers. To deploy the generated session bean,use the **cbejb** tool. The **mqaaejb** tool runsindependently of other EJB server (CB) tools.

To create a session bean for a particular MQSeries queue, do thefollowing:

1. Create a Java properties file that contains theseitems:
   - The message type specification--The property name must bemessageType, and its value must be either Inbound, Outbound, or InOut.If InOut is chosen, a pair of enterprise beans, instead of a single one, arecreated to accommodate paired inbound and outbound message queues. Hereis an example of this specification:

     ```
     messageType=Inbound
     ```

- A list of message field specifications--For each message field, theproperty name is the field name, and the property value is the fieldtype. Here is an example of this specification:

  ```
  bankName=java.lang.String

  accountNumber=int
  ```

  **Note:**

  Java class names in the type specifications must be the fully qualifiedpackage name.

2. Run the **mqaaejb** command with the following syntax:

   ```
   # mqaaejb -f propertiesFile -n baseBeanName [-p packageName]   [-i existingInboundBOInterfaceName]
   [-o existingOutboundBOInterfaceName]  [-c existingOutboundCopyName
   ```

   The -f and -n options are required. The *propertiesFile*specifies the name of the properties file created in Step 1, and the *baseBeanName* argument specifies the basename of the enterprise bean or beans to be generated. For example, ifthe base name is Account and the properties file specifies that it is for bothan inbound and an outbound message, then the **mqaaejb** commandgenerates session beans, related interfaces, and artifacts with the followingnames:

   AccountInboundBean

   AccountEJBObject

   AccountInboundEJBHome

   AccountOutboundBean

   AccountOutboundEJBObject

   AccountOutboundEJBHome

   AccountMsgTemplate

   The -p option specifies the package name of the enterprise bean; ifnot specified, the package name defaults to`mytest.ejb.mqaa`.

   Unless the -i option or the -o and -c options are specified, the**mqaaejb** command makes a mark for the **cbejb** command;later, when the **cbejb** command is run over the beans, it generatesthe required backing message BOs for the session beans. If you havealready created and tested MQSeries Application Adaptor-based BOs (followingthe procedure described in the MQSeries Application Adaptor Development Guide), you now need onlywrap them in session beans. You can specify the names of these BOs andthe Copy object to the **mqaaejb** command. The**mqaaejb** command then creates session beans that use the specifiedBOs. The names of these objects must be fully qualified. Forexample:

   ```
   mqaaejb -f mymsg.properties -n Account -i TextMessage::TMInbound \   -o TextMessage::TMOutbound -c
   TextMessageCopy::TMOutboundCopy
   ```

   You still must specify the base bean name with the -n option independentlyof the existing BOs. You also must provide a properties file; themessage format specified in this file must be consistent with the existingBOs. The correct mapping between the C++ field types in the BOs and theJava types in the properties file can be established by referring to the IDLC++/Java binding documentation.

   The following items are generated in the working directory on successfulcompletion of the **mqaaejb** command:

   - The Java source files (and the corresponding compiled class files) thatcompose the enterprise bean in the subdirectory corresponding to the packagename.
   - A JAR file containing the Java source files and compiled files thatcompose the enterprise bean.
   - An XML file containing the enterprise bean's deploymentdescriptor.

3. Run the **jetace** tool as follows to generate an EJB JAR file forthe enterprise bean:

   ```
   # jetace -f beanName.xml
   ```

4. Run the **cbejb** tool to deploy the enterprise bean contained inthe EJB JAR file. For more information, see Manually deploying an enterprise bean. When the **cbejb** command is complete,unless you are using existing BOs, you possibly need to follow the steps inthe MQSeries Application Adaptor Development Guide to modify the DOimplantation.

---

## Restrictions in the EJB server (CB) environment

The following restrictions apply when developing enterprise beans for the EJBserver (CB) environment:

- If you try to deploy an EJB JAR file that contains Java source files aswell as class files, or if you have JAR dependencies that include Java sourcecode in the JAR file, the deployment can fail with an I/O exception "Could notcompile." This is due to the javac compiler attempting to update anout-of-date.class file, with respect to the .java file included in the JARfile. To avoid this, ensure that the "export .java files"checkbox is not checked when you export your files to a JAR file from withinVisualAge for Java, or do not add the .java files to your JAR file whencreating it.
- Unqualified interface and exception names cannot be duplicated inenterprise beans. For example, thecom.ibm.ejs.doc.account.Account interfacemust not be reused in a package namedcom.ibm.ejs.doc.bank.Account. Thisrestriction is necessary because the EJB server (CB) tools generate enterprisebean support files that use the unqualified name only.
- Container-managed fields in entity beans must be valid for use in CORBAIDL files. Specifically, the variable names must use ISO Latin-1characters; they must *not* begin with an underscore character(_), they must *not* contain the dollar character ($), and they must*not* be CORBA keywords. Variables that have the same name butdifferent cases are not allowed. (For example, you cannot use thefollowing variables in the same class: *accountId* and*AccountId*. For more information on CORBA IDL, consult aCORBA programming guide.

Also, container-managed fields in entity beans must be valid Java types,but they *cannot* be of type ejb.javax.Handle or anarray of type EJBObject or EJBHome.

- The use of underscores (_) in the names of user-defined interfaces andexception classes is discouraged.
- Method names in the remote interface must *not* match methodnames in the Component Broker Managed Object Framework (that is, methods inthe IManagedServer::IManagedObjectWithCachedDataObject,CosStream::Streamable, CosLifeCycle::LifeCycleObject,and CosObjectIdentity::IdentifiableObject interfaces). Formore information on the Managed Object Framework, see the Component BrokerProgramming Guide. In addition, do not use underscores (_) atthe end of property or method names; this restriction prevents namecollision with queryable attributes in BO interfaces that correspond tocontainer-managed fields.
- The getUserTransaction method of the javax.ejb.EJBContextinterface (which is inherited by the SessionContext interface) returns anobject of type javax.transaction.UserTransaction rather thantype javax.jts.UserTransaction. While this is a deviationfrom the 1.0 version of the EJB Specification, the 1.1 versionof the EJB Specification requires that the getUserTransaction method return anobject of type javax.transaction.UserTransaction and drops therequirement to return objects of typejavax.jts.UserTransaction.
- The javax.ejb.SessionSynchronization interface is*not* supported.
- Entity beans with BMP that use Java Database Connectivity (JDBC) to accessa database cannot participate in distributed transactions because theenvironment does not support XA-enabled JDBC.
- The variables of the primary key class of a BMP entity bean must bepublic.
- The *run-as identity* and *access control* deploymentdescriptor attributes are not used.
- The remove method inherited by an enterprise bean's remote interface(from the javax.ejb.EJBObject interface) does not throw thejavax.ejb.RemoveException exception, even if the enterprisebean's corresponding ejbRemove() method throws this exception.This restriction is necessary because of the name conflict between the removemethod and the CORBACosLifeCycle::LifeCycleObject::remove method, which isinherited by all Component Broker managed objects.
- Single-threaded access to enterprise beans is enforced only if abean's transaction attribute is set to either TX_NOT_SUPPORTED orTX_BEAN_MANAGED. For other enterprise beans, access from differenttransactions is serialized, but serialized access from different threadsrunning under the same transaction is not enforced. Illegal callbacksfor enterprise beans deployed with the TX_NOT_SUPPORTED or TX_BEAN_MANAGEDtransaction attribute result in a java.rmi.RemoteExceptionexception being thrown to the EJB client.
- The session bean timeout attribute is *not* supported.
- The transaction attribute can be set only for the bean as a whole;the transaction attribute cannot be set on individual methods in abean.
- If a stateful session bean has the TX_BEAN_MANAGED transaction attributevalue, a method that begins a transaction must also complete that transaction(commit or roll back the transaction). In other words, a transactioncannot span multiple methods in a stateful session bean when used in the EJBserver (CB) environment.
- The TX_MANDATORY transaction attribute value must be used in entity beanswith container-managed persistence (CMP) that use HOD or ECI to access CICS orIMS applications. As a result, EJB clients that access these entitybeans must do so within a client-initiated one-phase commit transaction (CBsession service).
- The TX_NOT_SUPPORTED transaction attribute value is not supported forentity beans with CMP, because these beans must be accessed within atransaction.
- The TX_REQUIRES_NEW transaction attribute is *not* supported forJAR files that are in the EJB version 1.0 format. For JAR filesthat are in the EJB 1.1 format, the TX_REQUIRES_NEW transactionattribute is interpreted as TX_REQUIRED.
- For JAR files that are in the EJB 1.1 format, the TX_NEVERtransaction attribute is interpreted as TX_NOT_SUPPORTED.
- The TX_SUPPORTS transaction attribute is interpreted asTX_MANDATORY.
- The transaction isolation level attribute is *not*supported.
- When using thecom.ibm.ejb.cb.runtime.CBCtxFactory contextfactory, any of the default initial context factories (see Default context-to-finder associations), or an application-specific initial context factorygenerated by the **appbind** tool (see Application-specific contexts and the appbind tool), the javax.naming.Context.list andjavax.naming.Context.listBindings methods can return nomore than 1000 elements in the javax.naming.NamingEnumerationobject.
- C++ CORBA-based EJB clients are not supported.

# Tools for developing and deploying enterprise beans in the EJB server (AE) environment

There are two basic approaches to developing and deploying enterprise beans inthe EJB server (AE) environment:

- You can use one of the available integrated development environments(IDEs) such as IBM VisualAge(TM) for Java Enterprise Edition. IDE toolsautomatically generate significant parts of the enterprise bean code andcontain integrated tools for packaging and testing enterprise beans.VisualAge for Java is the recommended development tool for the EJB server (AE)environment. For more information on using VisualAge for Java, see Using VisualAge for Java.
- You can use the tools available in the Java Software Development Kit (SDK)and the Advanced Application Server. For more information, see Developing and deploying enterprise beans with EJB server (AE) tools.

**Note:**

Deployment and use of enterprise beans for the EJB server (AE) environmentmust take place on the Microsoft Windows NT(R) operating system, the IBMAIX(R) operating systems, or the Sun Microsystems Solaris operatingsystem.

For information on developing enterprise beans in the EJB server (CB)environment, see Tools for developing and deploying enterprise beans in the EJB server (CB) environment.

---

## Using VisualAge for Java

Before you can develop enterprise beans in VisualAge for Java, you must set upthe EJB development environment. You need to perform this setup taskonly once. This setup procedure directs VisualAge for Java to importall of the classes and interfaces required to develop enterprise beans.

After generating an enterprise bean, you complete its development byfollowing these general steps:

1. Implement the enterprise bean class.
2. Create the required abstract methods in the bean's home and remoteinterfaces by promoting the corresponding methods in the bean class to theappropriate interface.
3. For entity beans, do the following:
    a. Create any additional finder methods in the home interface by using theappropriate menu items.
    b. Create a finder helper interface, if required.
4. Create the EJB module and corresponding deployment descriptor.
5. Generate the deployment code for the bean.

VisualAge for Java contains a complete WebSphere Application Server runtime environment and a mechanism to generate a test client to test yourenterprise beans. For much more detailed information on developingenterprise beans in VisualAge for Java, refer to the VisualAge for Javadocumentation.

---

## Developing and deploying enterprise beans with EJB server (AE) tools

If you have decided to develop enterprise beans *without* an IDE, youneed at minimum the following tools:

- An ASCII text editor. (You can use also use a Java development toolthat does not support enterprise bean development.)
- The SDK Java compiler (**javac**) and Java Archiving tool(**jar**).
- The WebSphere Application Assembly Tool and the WebSphere AdministrativeConsole.

This section describes steps you can follow to develop enterprise beans byusing these tools. The following tasks are involved in the developmentof enterprise beans:

1. Ensure that you have installed and configured the prerequisite software todevelop, deploy, and run enterprise beans in the EJB server (AE)environment. For more information, see Installing and configuring the software for the EJB server (AE).
2. Set the CLASSPATH environment variable required by different components ofthe EJB server (AE) environment. For more information, see Setting the CLASSPATH environment variable in the EJB server (AE) environment.
3. Write and compile the components of the enterprise bean. For moreinformation, see Creating the components of an enterprise bean.
4. (*Entity beans with CMP only*) Create a finder helper interfacefor each entity bean with CMP that contains specialized finder methods (otherthan the findByPrimaryKey method). For more information, see Creating finder logic in the EJB server (AE).
5. Create an EJB module and corresponding deployment descriptor by using theApplication Assembly Tool. For more information, see Creating an EJB module.
6. (*Entity beans only*) Create a database schema to enable storageof the entity bean's persistent data in a database. For moreinformation, see Creating a database for use by entity beans.
7. Deploy the EJB module by using the Application Assembly Tool or theWebSphere Administrative Console. For more information, see theWebSphere InfoCenter and the online help available with the WebSphereAdministrative Console.

8. Install the EJB module into an EJB server (AE) and start the server byusing the WebSphere Administrative Console.

## Installing and configuring the software for the EJB server (AE)

You must ensure that you have installed and configured the followingprerequisite software products before you can begin developing enterprisebeans and EJB clients with the EJB server (AE):

- WebSphere Application Server Advanced Edition
- One or more of the following databases for use by entity beans withcontainer-managed persistence (CMP):
  - DB2
  - Oracle
  - Sybase
  - Informix
  - Microsoft SQL Server
  - InstantDB
- The Java Software Development Kit (SDK)

For information on the appropriate version numbers of these products andinstructions for setting up the environment, see the WebSphereInfoCenter.

## Setting the CLASSPATH environment variable in the EJB server (AE) environment

In addition to the classes.zip file contained in the SDK, the followingWebSphere JAR files must be appended to the CLASSPATH environment variable fordeveloping enterprise beans:

- ejs.jar
- ujc.jar
- *otherDeployedBean*.jar (if the enterprise bean usesanother enterprise bean). This is the deployed JAR file containing theenterprise bean being used by this enterprise bean.

For developing and running an EJB client, the following WebSphere JAR filesmust be appended to the CLASSPATH environment variable:

- ejs.jar
- ujc.jar
- servlet.jar (required by EJB clients that are servlets)
- *otherDeployedBean*.jar. This is the deployed JARfile containing the enterprise bean being used by this EJB client.

## Creating the components of an enterprise bean

If you use an ASCII text editor or a Java development tool that does notsupport enterprise bean development, you must create each of the componentsthat compose the enterprise bean you are creating. You must ensure thatthese components match the requirements described in Developing enterprise beans.

To manually develop a session bean, you must write the bean class, thebean's home interface, and the bean's remote interface. Tomanually develop an entity bean, you must write the bean class, thebean's primary key class, the bean's home interface, the bean'sremote interface, and if necessary, the bean's finderHelperinterface.After you have properly coded these components, use the Java compiler tocreate the corresponding Java class files. For example, because thecomponents of the example Account bean are stored in a specific directory, thebean components can be compiled by issuing the following command:

```
C:\MYBEANS\COM\IBM\EJS\DOC\ACCOUNT> javac *.java
```

This command assumes that the CLASSPATH environment variable contains allof the packages used by the Account bean.

## Creating finder logic in the EJB server (AE)

For the EJB server (AE) environment, the following finder logic is requiredfor each finder method (other than the findByPrimaryKey method) contained inthe home interface of an entity bean with CMP:

- The logic must be defined in a public interface named*Name*BeanFinderHelper, where *Name* is the name of theenterprise bean (for example, AccountBeanFinderHelper).
- The logic must be contained in a String constant named*findMethodName*WhereClause, where *findMethodName* is thename of the finder method. The String constant can contain zero or morequestion marks (?) that are replaced from left to right with the value of thefinder method's arguments when that method is invoked.

**Note:**

Encapsulating the logic in a String constant named*findMethodName*QueryString has been deprecated.

If you define the findLargeAccounts method shown in Figure 24, you must also create the AccountBeanFinderHelper interfaceshown in Figure 7.

**Figure 7. Code example: AccountBeanFinderHelper interface for the EJB server (AE)**

```
...public interface AccountBeanFinderHelper{     String findLargeAccountsWhereClause = "balance >
?";}
```

## Creating an EJB module

The WebSphere Application Server Application Assembly Tool can be used tocreate an EJB module. One or more enterprise beans can be placed in anEJB module. The tool automatically creates the required deploymentdescriptor for the module based on information specified by the user.

**Note:**

Before using the Application Assembly Tool, the WebSphere CommonConfiguration Model (WCCM) MetaObject Facility (MOF) JAR files must be addedto your CLASSPATH environment variable.

#### Using the Application Assembly Tool

To create an EJB module and corresponding deployment descriptor, use theCreate an EJB JAR wizard in the Application Assembly Tool. This wizardprompts you to specify the following information for each enterprise bean tobe included in the module:

- The enterprise bean class, home interface class, and remote interfaceclass.
- The bean type (entity or session), and associated attributes (such aspersistence management type and primary key class for entity beans).
- Any environment variables to be associated with the enterprisebean.
- References to another enterprise bean's home interface and toresource factories.
- References to security roles for the enterprise bean.

The wizard also prompts you to specify the following application assemblyinformation for the module itself:

- General properties of the EJB module, such as the location of class filesneeded for a client program to access the enterprise beans in the module andthe icons to be associated with the module.
- The deployable enterprise beans that the module will contain.
- Security roles used to access resources in the module.
- Transaction attributes for the enterprise bean methods.

Both bean and module information are used to create the deploymentdescriptor. See the WebSphere InfoCenter and the online help fordetails on how to use the Application Assembly Tool.

## Creating a database for use by entity beans

For entity beans with *container-managed persistence (CMP)*, you muststore the bean's persistent data in one of the supporteddatabases. The Application Assembly Tool automatically generates SQLcode for creating database tables for CMP entity beans. The tool namesthe database schema and table ejb.*beanName*beantbl, where*beanName* is the name of the enterprise bean (for example,ejb.accountbeantbl). If your CMP entity beans require complexdatabase mappings, it is recommended that you use VisualAge for Java togenerate code for the database tables. At run time, the WebSphereAdministrative Console displays a prompt asking whether you want to executethe generated SQL code that creates the database table.

For entity beans with *bean-managed persistence (BMP)*, you cancreate the database and database table by using the database tools or use anexisting database and database table. Because entity beans with BMPhandle the database interaction, any database or database table name isacceptable.

For more information on creating databases and database tables, consultyour database documentation and the online help for the WebSphereAdministrative Console.

# Appendix A. Changes for version 1.1 of the EJB specification

WebSphere Application Server supports version 1.1 of the EJBspecification. This appendix describes features that are new or havechanged in version 1.1 and discusses migration issues for enterprisebeans written to version 1.0 of the EJB specification.

---

## New and updated features

The following enterprise bean features are new or have changed for version1.1.

- Environmental dependencies for enterprise beans are now specified usingentries in a JNDI naming context. An instance of an enterprise beancreates a javax.naming.InitialContext object by invoking theconstructor with no arguments specified. It looks up the environmentnaming context by using the InitialContext object under the namejava:comp/env.
- Primary keys are handled differently in version 1.1 of the EJBspecification. Entity bean providers are not required to specify theprimary key class for entity beans with container-managed persistence (CMP),enabling the deployer to select the primary key fields when the bean isdeployed into a container.
- The deployment descriptor has enhanced support for applicationassembly.

---

## Migrating from version 1.0 to version 1.1

From the client's perspective, enterprise beans written to version1.1 of the EJB specification appear nearly identical to enterprisebeans written to version 1.0 of the specification. However, thefollowing EJB 1.1 changes do affect clients:

- Enterprise beans written to version 1.1 of the EJB specificationare registered in a different part of the JNDI namespace. For example,a client can look up the initial context of a version 1.0 enterprisebean in JNDI by using the **initialContext.lookup** method asfollows:

  ```
  initialContext.lookup("com/ibm/Hello")
  ```

  The JNDI lookup for the equivalent version 1.1 enterprise beanis:

  ```
  initialContext.lookup("java:comp/env/ejb/Hello")
  ```

- The UserTransaction object is obtained differently for enterprise beanswritten to version 1.1 of the EJB specification. Under version1.0, it was obtained as:

  ```
  initialContext.lookup("jta/UserTransaction")
  ```

  Under version 1.1, it is obtained as:

  ```
  initialContext.lookup("java:comp/UserTransaction")
  ```

- Because entity beans written to version 1.1 of the EJBspecification now support primitive primary keys (instead of having toencapsulate them in a primary key class), the client needs to look up theseprimitive keys directly. For example, a client can look up a primitivekey of the type java.lang.Integer as follows:

  ```
  accountHome.findByPrimaryKey(new Integer(5))
  ```

  Primary key classes are still supported, although their use forprimitive data types is deprecated.

From the application developer's perspective, the following changesneed to be made to make enterprise beans written to version 1.0 of theEJB specification compatible with version 1.1 of thespecification.

- All deployment descriptors must be converted to the XML format specifiedin version 1.1 of the EJB specification.

- In general, enterprise beans written to version 1.0 of the EJBspecification are compatible with version 1.1. However, you needto modify or recompile enterprise bean code in the following cases:
    - The return value of the ejbCreate method must be modified for all entitybeans with CMP. The ejbCreate method is now required to return the sametype as the primary key; the actual value returned must be null.These beans also must be recompiled. For more information, see Implementing the ejbCreate and ejbPostCreate methods
    - If the javax.jts.UserTransaction interface is used.This interface has been renamed tojavax.transaction.UserTransaction. Enterprise beans thatuse this interface must be modified to use the new interface name.There have also been minor changes to the exceptions thrown by thisinterface.
    - If the getCallerIdentity or isCallerInRole methods of thejavax.ejb.EJBContext interface are used. These methodswere deprecated because the javax.security.Identity class isdeprecated under the Java 2 platform.
    - If an entity bean uses the UserTransaction interface, which is notpermitted under version 1.1 of the EJB specification.
    - If an entity bean whose finder methods do not define the FinderExceptionin the methods' throws classes. Under version 1.1, thefinder methods of entity beans must define this exception.
    - If an entity bean uses the UserTransaction interface and implements theSessionSynchronization interface. Entity beans can neither use theUserTransaction interface nor implement the SessionSynchronization interfaceunder version 1.1.
    - If a stateful session beans implements the SessionSynchronizationinterface. This is not permitted under version 1.1.
    - If an enterprise bean violates any of the new semantic restrictionsdefined in version 1.1 of the EJB specification.
    - Throwing the javax.ejb.RemoteException exception from thebean implementations is deprecated in version 1.1. Thisexception should be replaced by the javax.ejb.EJBException or amore specific exception such as thejavax.ejb.CreateException. Thejavax.ejb.EJBException inherits from thejavax.ejb.RuntimeException and does not need to be explicitlydeclared in throws clauses.

      Declare the javax.ejb.RemoteException exception in the remoteand home interfaces, as required by RMI. Throwing this exceptiondirectly by the bean implementation is deprecated. However, it can bethrown by the container due to a system exception or by mapping an exceptionthrown by the bean implementation.

# Appendix B. Example code provided with WebSphere Application Server

This appendix contains information on the example code provided with theWebSphere Application Server for both Advanced Edition and EnterpriseEdition.

## Information about the examples described in the documentation

The example code discussed throughout this document is taken from a set ofexamples provided with the product. This set of examples is composed ofthe following main components:

- The Account entity bean, which models either a checking or savings bankaccount and maintains the balance in each account. An account ID isused to uniquely identify each instance of the bean class and to act as theprimary key. The persistent data in this bean is container managed andconsists of the following variables:
  - *accountId*--The account ID that uniquely identifies theaccount. This variable is of type long.
  - *type*--An integer that identifies the account as either asavings account (1) or a checking account (2). This variable is of typeint.
  - *balance*--The current balance of the account. Thisvariable is of type float.

  The major components of this bean are discussed in Developing entity beans with CMP.
- The AccountBM entity bean, which is nearly identical to the Account entitybean; however, the AccountBM bean implements bean-managedpersistence. This bean is not used by any other enterprise bean,application, or servlet contained in the documentation example set. Themajor components of this bean are discussed in Developing entity beans with BMP.
- The Transfer session bean, which models a funds transfer session thatinvolves moving a specified amount between two instances of an Accountbean. The bean contains two methods: the transferFunds methodtransfers funds between two accounts, the getBalance method retrieves thebalance for a specified account. The bean is stateless. Themajor components of this bean are discussed in Developing session beans.
- The CreateAccount servlet, which can be used to easily create new bankaccounts (and corresponding Account bean instances) with the specified accountID, account type, and initial balance. Although this servlet isdesigned to make it easy for you to create accounts and demonstrate the othercomponents in the example set, it also illustrates servlet interaction with anentity bean. This servlet is discussed in Developing servlets that use enterprise beans.
- The TransferApplication Java application, which provides a graphical userinterface that was built with the abstract windowing toolkit (AWT). Theapplication creates an instance of the Transfer session bean, which is thenmanipulated to transfer funds between two selected accounts or to get thebalance for a specified account. The TransferApplication codeimplements many of the requirements for using enterprise beans in an EJBclient. The parts of this application that are relevant to interactingwith an enterprise bean are discussed in Developing EJB clients.
- The TransferFunds servlet, which is a servlet version of theTransferApplication Java application. This servlet is provided so thatyou can compare the use of enterprise beans between a Java application and aJava servlet that basically are doing the same tasks. This documentdoes not discuss this servlet in any detail.

**Note:**

The example code in the documentation was written to be as simple aspossible. The goal of these examples is to provide code that teachesthe fundamental concepts of enterprise bean and EJB client

development.It is not meant to provide an example of how a bank (or any similar company)possibly approaches the creation of a banking application. For example,the Account bean contains a *balance* variable that has a type offloat. In a real banking application, you must not use a float type tokeep records of money; however, using a class likejava.math.BigDecimal or a currency-handling class within theexamples would complicate them unnecessarily. Remember this as youexamine these examples.

---

## Information about other examples in the EJB server (AE) environment

Table 4 provides a summary of the enterprise bean-specific examplesprovided with the EJB server (AE).

**Table 4. Examples available with the EJB server (AE)**

| Name | Bean types | EJB client types | Additional information |
|---|---|---|---|
| Hello | Stateless session | Java servlet | Very simple example of a session bean. |
| Increment | CMP entity | Java servlet | Very simple example of an entity bean. |

---

## Information about other examples in the EJB server (CB) environment

Table 5 provides a summary of the enterprise bean-specific examplesprovided with the EJB server (CB). or more information about theseexamples, see the README file that accompanies each example.

**Table 5. Examples available with the EJB server (CB)**

| Name | Bean types | EJB client types | Additional information |
|---|---|---|---|
| Hello | Stateless session | Java application | Very simple example of a session bean. |
| Calculator | Stateful session | Applet, ActiveX control | Demonstrates maintaining state information in a session bean. |
| Account | Stateful session, CMP entity, BMP entity | Servlet, Active X control | An Advanced Edition sample with a servlet client. One enterprisebean references another bean. |
| Card Game | Stateful session, CMP entity | Applet, ActiveX control | Demonstrates a session bean selecting entity beans with custom findermethods that use various types of queries. One enterprise beanreferences another bean. |
| Travel | Stateful session, BMP entity, CMP entity | Applet, ActiveX control | Demonstrates client-side transactions. An enterprise bean uses aPAA as a data source. One enterprise bean references anotherbean. |
| VisualAge for Java demo | CMP entity | | Demonstrates client-initiated transactions, inheritance, association andpolymorphic queries. One enterprise bean references anotherbean. |
| Big 3 | Stateless session, CMP entity | Multithreaded | Demonstrates enterprise beans written to version 1.1 of the EJBspecification. One enterprise bean references another bean. |
| Postcard | Stateless session | | Demonstrates enterprise beans that use Java Messaging Service (JMS)point-to-point messaging. |

| | | | |
|---|---|---|---|
| CORBA interoperability (policy wrapper) | BMP entity | | Demonstrates enterprise beans that communicate with C++ business objects(BO) and Java BOs (with a C++ client) that communicate with enterprisebeans. |
| JDBC AA | BMP entity | | Demonstrates how to use the CB Session service. An enterprise beanuses PAA as a data source. |

# Appendix C. Using XML in enterprise beans (CB Only)

**Note:**

This appendix applies to the EJB server (CB) environment only.Additionally, it applies only to creating XML deployment descriptors forenterprise beans that are written to version 1.0 of the EJBspecification. (The standard XML deployment descriptors are used forversion 1.1 enterprise beans.)

This appendix contains instructions for manually creating deploymentdescriptors for enterprise beans by using the extensible markup language(XML).

**Note:**

As an alternative to following these instructions, you can use VisualAge forJava to create XML deployment descriptors. See the VisualAge for Javaproduct documentation for details.

This appendix does not contain general information on creating or usingXML; for more information on XML, consult a commercially availablebook.

An XML file, which is a standard ASCII file, can be created manually or byusing the graphical user interface (GUI) of the **jetace** tool.Once created, the XML file can be used to create an EJB JAR file from thecommand line by using the **jetace** tool. For more information,see Creating an EJB JAR file for an enterprise bean.

An XML-based deployment descriptor must contain the following majorcomponents:

- Standard header and EJB JAR tags. For more information, see Creating the standard header and EJB JAR tags.
- The input file and output file tags. For more information, see Creating the input file and output file tags.
- Session bean or entity bean tag, depending on the type of bean for whichthe deployment descriptor is being generated. An XML file can containinstructions for generating an EJB JAR file with multiple enterprise beans ofall types. For more information, see Creating the entity bean tags and Creating the session bean tags.
- The tags used by all enterprise beans. For more information, see Creating tags used by all enterprise beans.

---

## Creating the standard header and EJB JAR tags

Every XML-based deployment descriptor must have the standard header tag,which defines the XML version and the standalone status of the XMLfile. For enterprise beans, these properties must be set to the valuesshown in Figure 121. Except for the header tag, which must be the firsttag in the file, the remaining content of the XML file must be enclosed inopening and closing EJB JAR tags.

**Figure 121. Code example: The standard header and EJB JAR tags**

```
<?xml version='1.0' standalone='yes' ?><ejb-JAR><!-- Content of the XML file -->...</ejb-JAR>
```

---

## Creating the input file and output file tags

The input file tag identifies the JAR or ZIP file or the directorycontaining the required components of one or more enterprise beans. Theoutput file tag identifies the EJB JAR file to be created; by default aJAR file is created, but you can force the creation of a ZIP file by adding a.zip extension to the output file name. The input and outputfiles for the example Account bean are shown in Figure 122.

**Figure 122. Code example: The input file and output file tags**

```
<?xml version='1.0' standalone='yes'
?><ejb-JAR><input-file>AccountIn.jar</input-file><output-file>Account.jar</output-file>...</ejb-JAR>
```

---

## Creating the entity bean tags

If you are creating a deployment descriptor for an entity bean, you mustuse an entity bean tag. The entity bean open tag must contain a dnameattribute, which must be set to the fully qualified name of the deploymentdescriptor associated with the entity bean.

Between the open and close entity bean tags, you must create the followingentity bean-specific attribute tags:

- <primary-key> -- Identifies the fully qualified name of theprimary key class for this entity bean.
- <re-entrant> -- Specifies whether the entity bean isre-entrant. This tag must contain a value attribute, which must be setto either `true` (re-entrant) or `false` (notre-entrant).
- <container-managed> -- Identifies the persistent variables in aCMP entity bean that are container managed. You must use a separate tagfor each persistent variable.

In addition to the entity bean-specific tags, you must create the tagsrequired by all enterprise beans described in Creating tags used by all enterprise beans.

Figure 123 shows the entity bean-specific tags for the example Accountbean.

**Figure 123. Code example: The entity bean-specific tags**

```
<?xml version='1.0' standalone='yes'
?><ejb-JAR><input-file>AccountIn.jar</input-file><output-file>Account.jar</output-file>...<entity-bean
dname="com/ibm/ejs/doc/account/Account.ser"><primary-key>com.ibm.ejs.doc.account.AccountKey</primary-key><re-entrant
value=false/><container-managed>accountId</container-managed><container-managed>type</container-managed><container-managed>balance</container-managed><!--Other
tags used by all enterprise beans--!>...</entity-bean>...</ejb-JAR>
```

## Creating the session bean tags

If you are creating a deployment descriptor for an session bean, you mustuse a session bean tag. The session bean open tag must contain a dnameattribute, which must be set to the fully qualified name of the deploymentdescriptor associated with the session bean. Between the open and closesession bean tags, you must also create the following session bean attributetags:

- <session-timeout> -- Defines the idle timeout in secondsassociated with the session bean.
- <state-management> -- Identifies the type of session bean:`STATELESS_SESSION` or `STATEFUL_SESSION`.

In addition to the session bean-specific tags, you must create the tagsrequired by all enterprise beans described in Creating tags used by all enterprise beans.

Figure 124 shows the session bean tags for the example Transferbean.

**Figure 124. Code example: The session bean-specific tags**

```
<?xml version='1.0' standalone='yes'
?><ejb-JAR><input-file>TransferIn.jar</input-file><output-file>Transfer.jar</output-file>...<session-bean
dname="com/ibm/ejs/doc/transfer/Transfer.ser"><session-timeout>0<\session-timeout>
<state-management>STATELESS_SESSION<\state-management><!--Other tags used by all enterprise
beans--!>...</session-bean>...</ejb-JAR>
```

## Creating tags used by all enterprise beans

The following tags are used by all types of enterprise beans. Thesetags must be placed between the appropriate set of opening and closing sessionor entity bean tags in addition to the tags that are specific to those typesof beans.

- <remote-interface> -- Identifies the fully qualified name of theenterprise bean's remote interface.
- <enterprise-bean> -- Identifies the fully qualified name of theenterprise bean's bean class.
- <JNDI-name> -- Identifies the JNDI home name of the enterprisebean.
- <transaction-attr> -- Defines the transaction attribute for theentire enterprise bean. This attribute can also be set for anindividual bean method. The valid values are `TX_MANDATORY`,`TX_NOT_SUPPORTED`, `TX_REQUIRES_NEW`,`TX_REQUIRED`, `TX_SUPPORTS`, and`TX_BEAN_MANAGED`. For more information on the meaning of andrestrictions on these values, see Setting the transaction attribute.
- <isolation-level> -- Defines the transactional isolation levelattribute for the entire enterprise bean. This attribute can also beset for an individual bean method. The valid values, which must be setby using a value attribute within the open tag, are `SERIALIZABLE`,`REPEATABLE_READ`, `READ_COMMITTED`, and`READ_UNCOMMITTED`. For more information on the meaning of andrestrictions on these values, see Setting the transaction isolation level attribute.
- <run-as-mode> -- Defines the run-as mode attribute for the entireenterprise bean. This attribute can also be set for an individual beanmethod. The valid values, which must be set by using a value attributewithin the open tag, are `CLIENT_IDENTITY`,`SYSTEM_IDENTITY`, and `SPECIFIED_IDENTITY`. For moreinformation on the meaning of these values, see Setting the security attribute in the deployment descriptor.
- <run-as-id> -- Defines the run-as identity attribute for theentire enterprise bean. This attribute can also be set for anindividual bean method. This attribute is not used with the EJB serverenvironments contained in WebSphere Application Server.
- <method-control> -- Identifies individual bean methods withtransaction or security attributes that are different from the attributevalues for the entire bean.
- <dependency> -- Identifies the fully qualified names of classes onwhich this enterprise bean is dependent.
- <env-setting> -- Identifies environment variables (and theirvalues) required by the enterprise bean. The environment variable nameis specified with a name attribute, while the environment variable value isplaced between the open and close tags.

Figure 125 shows the enterprise bean tags for the example Transferbean. A similar set is required by the Account bean.

**Figure 125. Code example: The tags used for all enterprise beans**

```
<?xml version='1.0' standalone='yes'
?><ejb-JAR><input-file>TransferIn.jar</input-file><output-file>Transfer.jar</output-file>...<session-bean
dname="com/ibm/ejs/doc/transfer/Transfer.ser"> <!--Session bean-specific tags
--!>...<remote-interface>com.ibm.ejs.doc.transfer.Transfer</remote-interface><enterprise-bean>com.ibm.ejs.doc.transfer.TransferBean</enterprise-bean><JNDI-name>Transfer
</JNDI-name><transaction-attr value="TX_REQUIRED"/><isolation-level value="SERIALIZABLE"/> <run-as-mode
value="CLIENT_IDENTITY"/><dependency>com/ibm/ejs/doc/account/InsufficientFundsException.class</dependency>...<env-setting
name="ACCOUNT_NAME">Account<env-setting>...</session-bean>...</ejb-JAR>
```

If you want to override the enterprise bean-wide transaction or securityattribute for particular method in that bean, you must use the<method-control> tag. Between the open and close tags, you mustidentify the method with the <method-name> tag and the method'sparameter types by using the <parameter> tag. In addition, thefollowing tags can used to identify those attribute values that are differentin the method from the enterprise bean as a whole:<transaction-attr>, <isolation-level>, <run-as-mode>, and<run-as-id>.

For example, the XML shown in Figure 126 is required to override the transaction attribute of theTransfer bean (TX_REQUIRED) in the getBalance method to TX_SUPPORTED.Because only the transaction attribute is overridden, the method automaticallyinherits the values of the <isolation-level> and <run-as-mode> tags fromthe Transfer bean.

**Figure 126. Code example: Method-specific tags**

```
<?xml version='1.0' standalone='yes'
?><ejb-JAR><input-file>TransferIn.jar</input-file><output-file>Transfer.jar</output-file>...<session-bean
dname="com/ibm/ejs/doc/transfer/Transfer.ser"> <!--Session bean-specific tags --!>...<transaction-attr
value="TX_REQUIRED"/><isolation-level value="SERIALIZABLE"/> <run-as-mode
value="CLIENT_IDENTITY"/>...<method-control><method-name>getBalance</method-name><parameter>long</parameter><transaction-attr
value="TX_SUPPORTED"/></method-control></session-bean>...</ejb-JAR>
```

# Appendix D. Extensions to the EJB Specification

This appendix briefly discusses functional extensions to the EJBSpecification that are available in the EJB server environments contained inWebSphere Application Server. These extensions are specific toWebSphere Application Server and use of these features is supported only withVisualAge for Java, Enterprise Edition. For information on implementingthese features, consult your VisualAge for Java documentation.

---

## Access beans

*Access beans* are Java components that adhere to the SunMicrosystems JavaBeans(TM) Specification and are meant to simplify developmentof EJB clients. An access bean adapts an enterprise bean to theJavaBeans programming model by hiding the home and remote interfaces from theaccess bean user (that is, an EJB client developer). Access beans aresupported in both the Advanced Edition and Component Broker EJBenvironments.

There are three types of access beans, which are listed in ascending orderof complexity:

- Java bean wrapper--Of the three types of access beans, a Java beanwrapper is the simplest to create. It is designed to allow either asession or entity enterprise bean to be used like a standard Java bean and ithides the enterprise bean home and remote interfaces from you. EachJava bean wrapper that you create extends thecom.ibm.ivj.ejb.access.AccessBeanclass.
- Copy helper--A copy helper access bean has all of the characteristicsof a Java bean wrapper, but it also incorporates a single copy helper objectthat contains a local copy of attributes from a remote entity bean. Auser program can retrieve the entity bean attributes from the local copyhelper object that resides in the access bean, which eliminates the need toaccess the attributes from the remote entity bean.
- Rowset--A rowset access bean has all of characteristics of both theJava bean wrapper and copy helper access beans. However, instead of asingle copy helper object, it contains multiple copy helper objects.Each copy helper object corresponds to a single enterprise beaninstance.

VisualAge for Java provides a SmartGuide to assist you in creating orediting access beans.

---

## Associations between enterprise beans

In the EJB server environment, an association is a relationship that existsbetween two CMP entity beans. There are three types ofassociations: one-to-one and one-to-many. In a one-to-oneassociation, a CMP entity bean is associated with a single instance of anotherCMP entity bean. For example, an Employee bean could be associated withonly a single instance of a Department bean, because an employee generallybelongs only to a single department.

In a one-to-many association, a CMP entity bean is associated with multipleinstances of another CMP entity bean. For example, a Department beancould be associated with multiple instances of an Employee bean, because mostdepartments are made up of multiple employees.

The Association Editor is used to create or edit associations between CMPentity beans in VisualAge for Java.

---

## Inheritance in enterprise beans

In Java, *inheritance* is the creation of a new class from anexisting class or a new interface from an existing interface. The EJBserver environment permits two forms of inheritance: standard classinheritance and EJB inheritance. In standard class inheritance, thehome interface, remote interface, or enterprise bean class inherits propertiesand methods from base classes that are not themselves enterprise bean classesor interfaces.

In enterprise bean inheritance, by comparison, an enterprise bean inheritsproperties (such as CMP fields and association ends), methods, andmethod-level control descriptor attributes from another enterprise bean thatresides in the same group.

VisualAge for Java provides a SmartGuide to assist you in implementinginheritance in enterprise beans.

# 6.6.4: Administering EJB containers (overview)

A container configuration provides information about an enterprisebean container. The administrator can specify several properties toaddress basic questions about the container location and behavior.

## Specifying the server in which the container will reside

Each enterprise bean container resides in a particular application server.

When the administrator adds a new container to the WebSphere administrativedomain, he or she must associate the container with a particular server (alsoknown as the container's parent).

An application server can host multiple containers.

## Specifying how beans in the container will get database connections

Every container can support the two main bean types, session beans and entitybeans:

- Entity beans require database connections because they store permanent data.
- Session beans do not *require* database access, though they can obtain it indirectly (as needed) by accessing entity beans.

A data source is an administrative resource that defines a pool of database connections. Servlets and enterprise beans use data sources to obtain database connections.

When configuring a container, the administrator can specify a default data source for the container. This data source will be the default data source used by any entity beans installed in the container that use container managed persistence (CMP).

When configuring a CMP entity bean, the administrator can specify which data source the container must use for managing the persistent state of the entity bean. If the administrator specifies a data source for an individual CMP entity bean then this data source will override any data source specified on the container.

Specifying a default data source is optional if each CMP entity bean in the container has a data source specified in it configuration. If a default data source is not specified and a CMP entity bean is installed in that container without specifying a data source for that bean then it will not be possible to start that CMP entity bean.

The default data source for a container is secure. When specifying it, the administrator must provide the user ID and password for accessing the data source.

## Specifying how the container will manage cached bean instances

Each container keeps a cache of bean instances for ready access. The WebSphere administrator specifies settings governing the cache size and a policy for removing unused items from the cache.

## Specifying where the container will passivate beans to make room in its cache

A container can *passivate* session beans to make room in its cache. The container saves a serialized session bean to a file. It restores the bean tothe cache when more room is available.

The WebSphere administrator specifies a passivation directory inwhich to keep the files.

# 6.6.4.0: EJB container properties

**Cache absolute limit**

Specifies the maximum number of bean instances permitted in the cache by the container cache manager. The container will fail to allocate new bean instances when the total number of active beans reaches this limit.

This value must be a positive integer.

**Cache clean-up interval**

Specifies the interval at which the container attempts to remove unused items from the cache to reduce the total number of items in the cache to the value of the Cache preferred limit property.

The cache manager tries to maintain some unallocated entries that can be quickly allocated as needed. A background thread attempts to free some entries while ensuring that some unallocated entries are maintained. If the thread runs while the application server is idle, then when the application server needs to allocate new cache entries, it does not pay the performance cost of removing entries from the cache.

In general, increase this parameter as the cache size increases.

This value must be a positive integer specified in milliseconds.

**Cache preferred limit**

Specifies a soft limit for the number of bean instances the container attempts to retain in the cache. The cache manager will use this value as a trigger to start discarding unused entries from the cache. See the cache clean-up interval description for details about the cleanup mechanism.

If necessary, the number of enterprise bean instances in the cache can increase to the value specified by the Cache absolute limit property. The difference between the Cache preferred limit and the Cache absolute limit can be thought of as the "surge capacity" for the container -- that is, the ability of the container to handle a spike in requests without having to passivate beans.

This value must be an integer less than or equal to the value of the Cache absolute limit.

**Cache size**

Specifies the number of buckets in the cache hash table.

If you change this value, change the Cache absolute limit property to correspond. For example, if you change the cache size to 3000, change the cache absolute limit to 3000, unless for some reason you do not want all of the available cache to be used.

This value must be a positive integer.

**Current State**

Indicates the state the container is currently in. The next time itis started, it will try to change to its desired state setting.

**Data Source**

Specifies the data source the container should use for the purpose of enterprise bean persistence.

**Data Source in use**

Specifies the data source currently in use.

**Desired state**

Indicates the state the container should have the next time it is started.

**EJB Container name**

Specifies a name for the container. The name must be unique within the application server that contains it.

**Name**

Indicates the name of the container

**Passivation directory**

Specifies the name of a directory where the container saves the persistent state of passivated session beans.

Session beans are passivated when the container needs to reclaim space in the bean cache. At passivation time, the container serializes the bean instance to a file in the passivation directory and discards the instance from the bean cache.

If, at a later time, a request arrives for the passivated bean instance, the container retrieves it from the passivation directory, deserializes it, returns it to the cache, and dispatches the request to it.

If any of these steps fail (for example, if the bean instance is no longer in the passivation directory), then the method invocation fails.

**Password**

Specifies the password for accessing the container's data source.

**Start time**

Indicates the time at which the container was most recently started.

**State**

Indicates the state the container is currently in. The next time itis started, it will try to change to its desired state setting.

**User ID**

Specifies the user ID for accessing the container's data source.

**User ID in use**

Specifies the user ID currently in use.

# 6.6.4.1: Administering enterprise bean containers with the Java administrative console

This article extends article 6.6.4 (the overview of administering enterprise bean containers) with information specific to the Java console.

The table answers the most basic questions. See the Related informationfor links to detailed instructions and resource properties.

| | |
|---|---|
| Does the console provide full functionality for administering this resource? | Yes |
| How is this resource representedin the console tree views? | The Type tree contains a Containers folderobject.<br><br>The Topology tree can contain zero or moreexisting containers. Their names vary;they are supplied by the administrator.<br><br>Use the View menu on the console menu bar to toggle between tree views. |
| Any task wizards for manipulatingthis resource? | Not directly, though a container can be configuredas part of the tasks (on the console menu bar):<br><br>Console -> Task -> Create application server |

# 6.6.4.1.1: Configuring new EJB containers with the Java administrative console

Use menus on resources in the Topology and Type treesto configure new containers (see Related information forinstructions).

# 6.6.4.1.4: Tuning containers with the Java administrative console

The IBM Redbook SG24-5657-00 is a recommended source of container tuning guidelines. Some tips are excerpted and included in the container propertyhelp (section 6.4.1.4.1.4).

Although it provides figures for Version 3.0x, the performance tuning Redbook discusses many tuning principles that can be anticipated to apply to Version 3.5. Specific performance numbers and suggested setting values, which have not been verified with Version 3.5, might differ.

The book provides guidelines such as the cache size estimation metric that follows.

## Estimating the cache size

Sizing the cache involves estimating the working set size for the concurrent load to which you expect the application server to be subjected.

To determine a rough approximation of the required value for this property, multiply the number of beans active in any given transaction by the total number of concurrent transactions expected. Then add the number of active entity bean instances.

For example, an EJB model:

- with 1 stateful session bean
- and 5 entity beans
- accessed by 200 concurrent clients

would have 1200 active beans:

```
200 x 1 stateful session beans, plus 200 x 5 entity beans
```

In this case, set your cache to be equal to or greater than 1200.

Given the high cost of passivating a bean when the container cache absolutelimit is reached, set the container cache and the container cache absolute limit to be larger than the expected load, ratherthan setting these values too low.

# 6.6.4.4: Property files pertaining to containers

The container properties are in file:

- *nameservice.config*

This file is located in directory:

`<WebSphere/Appserver>/properties`

The following entries in the ***nameservice.config*** file are used to administer containers:

| | |
|---|---|
| Container.implClass | identifies the name of the container |
| Container.dbUrl | specifies the data source the container should use for the purpose of enterprisebean persistence |
| Container.jarFileDirectory | specifies the name of the directory where containers and persistent states are located |

The container properties are in file:

- *nameservice.config*

This file is located in directory:

`<WebSphere/Appserver>/properties`

The following entries in the ***nameservice.config*** file are used to administer containers:

| | |
|---|---|
| Container.implClass | identifies the name of the container |
| Container.dbUrl | specifies the data source the container should use for the purpose of enterprisebean persistence |
| Container.jarFileDirectory | specifies the name of the directory where containers and persistent states are located |

# 6.6.5: Administering enterprise beans (overview)

Because enterprise beans are packaging into JAR files and code is generatedfor deployment, long before they are installed into the application server runtime,most of the enterprise bean and EJB module administration applies to the EJB container level.

# 6.6.5.0: Enterprise bean properties

**Create table**

Specifies whether to create a table in the data source for persistent data.

**Create table in use**

Indicates whether a table was created in the data source for persistent data.

**Current state**

Indicates the state the enterprise bean is currently in. The next time it is started, it will try to change to its desired state setting.

**Database access**

Specifies whether the persistent data of entity beans is cached in memory across transactions.

By default, a container loads persistent data for entity beans at the start of each transaction. If you use cached entity beans, you are not guaranteed the correctness of bean data due to updates made by other processes.

Use cached entity beans only if you know that the container has exclusive access to the database used by the entity bean (and therefore has the only copy of a bean's persistent state), or that the bean's data is accessed read-only at all times.

**Database access in use**

Indicates whether the persistent data of entity beans is being cached in memory across transactions.

**Data source**

Specifies the data source in which to keep persistent data.

**Data source in use**

Indicates the data source now in use.

**Deployment descriptor**

Specifies the full path name of the deployment descriptor file to use the next time the server is started.

**Desired state**

Indicates the state the enterprise bean should have the next time it is started.

**Find for update**

Specifies whether the container should get an exclusive lock on the enterprise bean when the "find by primary key" method is involved. The setting will take effect the next time the application sever hosting the enterprise bean is started.

This setting is useful for avoiding deadlock in the database. Deadlock can occur when two transactions execute find methods, and then update methods, on the same enterprise bean. The find method grants a shared lock on the enterprise bean, but the update method attempts to get an exclusive lock on the enterprise bean, resulting in deadlock.

**Find for update in use**

Indicates the current value of the Find for update property.

**JAR file**

Specifies the full path name of the JAR file to use the next time the server is started.

**JAR file in use**

Indicates the full path name of the JAR file now being used by the server for the enterprise bean.

**Maximum pool size**

Specifies the maximum number of pooled instances the container of the enterprise bean can have on behalf of the bean.

**Maximum pool size in use**

Indicates the current value of the Maximum pool size property.

**Minimum pool size**

Specifies the minimum number of pooled instances the container of the enterprise bean can have on behalf of the bean.

**Minimum pool size in use**

Indicates the current value of the Minimum pool size property.

**Name**

Specifies a name for the enterprise bean. The name must be unique within the administrative domain.

**Password**

Specifies the password for accessing the data source.

**Start time**

Indicates the time that the enterprise bean was started or restarted.

- Class: Runtime
- Data Type:

**State**

Indicates the state the enterprise bean is currently in. The next time itis started, it will try to change to its desired state setting.

**User ID**

Specifies the user ID for accessing the data source.

**User ID in use**

Specifies the user ID currently being used to access the data source.

# 6.6.5.1: Administering enterprise beans with the Java administrative console

This article extends article 6.6.5 (the overview of administering enterprise beans) with information specific to the Java console.

The table answers the most basic questions. See the Related informationfor links to detailed instructions and resource properties.

| | |
|---|---|
| Does the console provide full functionality for administering this resource? | Yes |
| How is this resource represented in the console tree views? | The Type tree contains a Enterprise Beans folder object.<br><br>The Topology tree can contain zero or more existing enterprise beans. Their names vary;they are supplied by the administrator.<br><br>Use the View menu on the console menu bar to toggle between tree views. |
| Any task wizards for manipulating this resource? | On the console menu bar:<br><br>Console -> Task -> Deploy enterprise beans |

# 6.6.5.1.1: Configuring new enterprise beans

For any application server product, a procedure is required to put adeveloped enterprise bean onto an application server where it can be madeavailable to users.

This section outlines the procedure for the WebSphere Application Serverproduct, from the administrator's point of view.

1. The enterprise bean developer writes and compiles the enterprise bean components. The developer packages the components and a deployment descriptor into an EJB JAR file containing a manifest.

   For entity beans (BMP or CMP), the developer generates the database tables the beans will use to store their data.
2. The developer transfers the JAR file to the WebSphere administrator, or informs the administrator of its location on a machine in the WebSphere administrative domain.

   The developer tells the administrator whether the JAR file has been deployed.

   A developer using VisualAge for Java can deploy the JAR file before giving the file to the administrator. A deployed JAR file consists of the EJBHome and EJBObject classes, persistor and finder classes, and stub and skeleton files.

   Otherwise, the administrator makes a note to deploy the JAR file while installing it in the WebSphere administrative domain.

   There are special considerations for deploying entity beans with container-managed persistence (CMP) and any enterprise beans with EJB inheritance.
3. The administrator installs the JAR file in the administrative domain, deploying the JAR file if necessary.

   Installing an enterprise bean refers to the process of placing the bean in a runtime environment comprised of an application server and enterprise bean container.

   During this step, the administrator can optionally edit the bean deployment descriptor.
4. If the beans in the JAR file reference classes outside of the JAR file, the administrator adds the referenced classes to the CLASSPATH environment variable of the machine on which the beans are installed.

   The bean JAR file itself is automatically added to the CLASSPATH when the administrator installs the bean JAR file in the WebSphere domain. If the referenced classes are contained in the JAR file, no action is required.
5. The administrator or developer prepares the enterprise bean for workload management (recommended).

   This step is not required for JAR files deployed in VisualAge for Java.
6. The administrator starts the enterprise bean, perhaps after adding it to an enterprise application.
7. After changing the enterprise bean, the developer provides a replacement JAR file to the administrator. The administrator adds the file to the WebSphere administrative domain.
   - If the administrator treats the JAR file as a new one, the administrator can install the deployed file into a running application server without having to stop the server and start it again.

     The administrator should delete the old JAR file from the WebSphere directories so that there is no chance it will be used.
   - If the administrator treats the JAR file as a replacement for an existing one, the administrator must stop the application server on which the bean is running and start it again after installing the

replacement JAR file.

# Special deployment considerations

- Deploying entity beans with CMP
- Deploying enterprise beans with EJB inheritance

## Considerations for deploying entity beans with CMP

If you are using CMP entity beans that do not rely on a particular databaseconfiguration (that is, the beans are not storing data in legacy applications or inexisting database tables) you can use the WebSphere Administrative Console toautomatically create the deployed JAR file and the corresponding database table.

If you are using CMP entity beans for a legacy application (or the beans are from athird party vendor), you must use VisualAge for Java to create the deployed JAR file. Youcan then create (install) the bean by using the WebSphere Administrative Console.

It is strongly recommended that you use VisualAge for Java for deploying beans used inlegacy applications or beans that require complex mappings to a database table. If you usethe automatic deployment process in the console, the order and names of the columns in thegenerated table are not guaranteed to match the table configuration needed by the legacyapplication. (The console deployment process makes certain assumptions about the order ofcontainer-managed fields.)

If you decide to use automatic deployment within the console, but want to manuallycreate the database table, note the following:

- The name of the database table must follow the convention EJB**.***beanname*BeanTbl. There is a 14-character limit on the length of table names in DB2.
- The primary key fields must appear first, and the column headings in the database must match the name and order of the fields as they appear in the deployment descriptor.

An entity bean with CMP must be associated with the name of a data source. Adata source specifies a database name, Uniform Resource Locator (URL), networkprotocol, and location in the Java Naming and Directory Interface (JNDI) namespace.

A data source also references a JDBC driver, used to locate thedriver's JAR file on the node. When you create the bean, you are prompted tosupply the name of this data source.

## Considerations for deploying JAR files with EJB inheritance

Consider the following when deploying JAR files with EJB inheritance:

- Enterprise beans that participate in an inheritance hierarchy must be deployed in a single JAR file, and you must install and uninstall the inheritance hierarchy as a unit.
- You must modify the JNDI name of the home for each enterprise bean within the hierarchy. The JNDI name of each bean in the hierarchy must be unique within its container.

# 6.6.5.4: Property files pertaining to enterprise beans

The enterprise bean properties are in file:

- *admin.config*

The *admin.config* file is located in directory,
`<WebSphere/Appserver>/bin`

The following entries in the admin.config file apply to enterprise beans:

| | |
|---|---|
| com.ibm.ejs.sm.adminServer.nameServiceJar | name of service bean jar file |
| com.ibm.ejs.sm.adminServer.dbUrl | URL for JDBC access |
| com.ibm.ejs.sm.adminServer.dbDriver | classname of JDBC driver |
| com.ibm.ejs.sm.adminServer.connectionPoolSize | size of database connection pool |
| com.ibm.ejs.sm.adminServer.dbPassword | password for database access |
| com.ibm.ejs.sm.adminServer.dbUser | user ID for database access |

The enterprise bean properties are in file:

- *admin.config*

The *admin.config* file is located in directory,
`<WebSphere/Appserver>/bin`

The following entries in the admin.config file apply to enterprise beans:

| | |
|---|---|
| com.ibm.ejs.sm.adminServer.nameServiceJar | name of service bean jar file |
| com.ibm.ejs.sm.adminServer.dbUrl | URL for JDBC access |
| com.ibm.ejs.sm.adminServer.dbDriver | classname of JDBC driver |
| com.ibm.ejs.sm.adminServer.connectionPoolSize | size of database connection pool |
| com.ibm.ejs.sm.adminServer.dbPassword | password for database access |
| com.ibm.ejs.sm.adminServer.dbUser | user ID for database access |