IBM VisualAge® for Java™, Version 3.5

EJB Development Environment

Note!
Before using this information and the product it supports, be sure to read the general information under **Notices**.

Edition notice

This edition applies to Version 3.5 of IBM VisualAge for Java and to all subsequent releases and modifications until otherwise indicated in new editions.

© Copyright International Business Machines Corporation 1998, 2000. All rights reserved.
US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

© Copyright IBM Corp. 1998, 2000

Chapter 1. EJB Development Environment

Overview of the EJB Development Environment

The IBM VisualAge® for Java™ *EJB Development Environment* is a specialized environment that you can use to develop and test enterprise beans that conform to the distributed component architecture defined in the Sun Microsystems Enterprise JavaBeans™ (EJB) specification. If you are not familiar with enterprise beans or related EJB technology, a brief description of key EJB concepts is found in the topic "EJB architecture." A more complete description of EJB technology is found in the Sun Microsystems EJB 1.0 specification available at the following Web site:

http://java.sun.com/products/ejb/docs10.html

The EJB Development Environment consists of multiple tools:

- An EJB page in the VisualAge for Java Workbench
- Tools for creating enterprise beans and access beans
- Tools for building data persistence into enterprise beans
- Tools for generating deployed code
- Tools for verifying enterprise bean code
- Tools for testing enterprise beans

EJB page
All of the EJB Development Environment tools are accessible from the EJB page of the VisualAge for Java Workbench. The EJB page is the heart of the EJB Development Environment. This is where your EJB groups and individual enterprise beans reside, and it is where you accomplish all of your enterprise bean development and testing activities.

Creating enterprise beans and access beans
The EJB Development Environment provides tools to help you create enterprise beans (either with or without inheritance), such as session beans, container-managed persistence (CMP) entity beans, and bean-managed persistence (BMP) entity beans.

Tools are also provided to create access beans and other EJB elements, such as associations. Access beans are Java bean wrappers for enterprise beans, which are typically used by client programs, such as Java ServerPages (JSP) files, servlets, and sometimes even other enterprise beans. Detailed information about access beans is found in the topic "Access beans."

There are also tools that allow you to accomplish complementary enterprise bean development activities, such as writing and editing business logic, importing or exporting enterprise beans, and maintaining both your enterprise bean source code and generated code using the built-in team and versioning capabilities of VisualAge for Java.

Building data persistence into enterprise beans
The EJB Development Environment provides tools to help you map entity enterprise beans to back-end data stores, such as relational databases. There is support for top-down, bottom-up, and meet-in-the-middle development. You can create EJB groups from existing schemas or Persistence Builder models. You can

© Copyright IBM Corp. 1998, 2000

also create schemas and maps from existing EJB groups. Additional information is found in the topic "Approaches for mapping enterprise beans to database tables."

Generating deployed code
EJB Development Environment tools are provided to set deployment descriptor and control descriptor properties for your enterprise beans and to generate the deployed classes that allow your beans to operate on an EJB server. The tool used to generate the deployed code is integrated with the VisualAge for Java generation options, so you can simply select individual enterprise beans or EJB groups as input and then select a menu item to automatically generate the deployed code. These tools mask the complexity normally associated with creating deployed classes, such as generating RMI-over-IIOP stubs and EJB container-specific deployed code.

The tools support session beans, CMP entity beans, and BMP entity beans. They also allow you to create relational database tables for CMP entity beans. Once the deployed code has been generated, you can export your enterprise beans to a JAR file for installation on an EJB server, such as the WebSphere™ Application Server.

Verifying enterprise bean and access bean code
The EJB Development Environment automatically and seamlessly verifies that your enterprise bean code is consistent and that it conforms to the rules defined by the Sun EJB specification. Code verification occurs whenever an enterprise bean or its properties are changed. If any errors are detected, an error or warning icon appears beside the enterprise bean and a message appears in the status bar at the bottom of the EJB page.

The EJB Development Environment also automatically verifies that access beans are constructed correctly and that they are consistent with their associated enterprise beans. Code verification occurs whenever you create or edit access beans.

Testing enterprise beans
The EJB Development Environment incorporates a test environment that you can use to test run your enterprise beans before installing them on a production EJB server. The EJB Development Environment contains the run-time environment of the WebSphere Application Server. When you test an enterprise bean in this run-time environment, it is similar to testing the enterprise bean in a production WebSphere Application Server environment.

The test environment provides an EJB server and a user interface that allows you to test and debug your enterprise beans. You can create and manage server configurations, which are instances of the EJB server running inside the VisualAge for Java Virtual Machine. For example, you can configure the EJB server with a set of enterprise beans and then set breakpoints in your enterprise bean code. This allows you to leverage the full debugging capabilities of VisualAge for Java to debug your enterprise beans.

A test client is also provided to help you test your enterprise beans. The test client features its own user interface and allows you to test the individual methods in the home and remote interfaces of each enterprise bean.

RELATED CONCEPTS

EJB architecture
Inheritance and association

Approaches for mapping enterprise beans to database tables
Access beans
Limitations

RELATED TASKS

Using the EJB Development Environment: overview

EJB architecture

This topic provides a high-level overview of the distributed component architecture defined in the Sun Microsystems Enterprise JavaBeans (EJB) specification. A more complete description of this architecture is found in the EJB 1.0 specification available at the following Web site:

http://java.sun.com/products/ejb/docs10.html

Enterprise beans
An *enterprise bean* is a nonvisual component of a distributed, transaction-oriented enterprise application. Enterprise beans are typically deployed in EJB containers and run on EJB servers. You can customize them by changing their deployment descriptors and you can assemble them with other beans to create new applications.

There are two types of enterprise beans: *session beans* and *entity beans*.

Session beans are non-persistent enterprise beans. They can be stateful or stateless. A *stateful session bean* acts on behalf of a single client and maintains client-specific session information (called conversational state) across multiple method calls and transactions. It exists for the duration of a single client/server session. A *stateless session bean*, by comparison, does not maintain any conversational state. Stateless session beans are pooled by their container to handle multiple requests from multiple clients.

Entity beans are enterprise beans that contain persistent data and that can be saved in various persistent data stores. Each entity bean carries its own identity. Entity beans that manage their own persistence are called *bean-managed persistence (BMP) entity beans*. Entity beans that delegate their persistence to their EJB container are called *container-managed persistence (CMP) entity beans*.

Enterprise beans provide several benefits for application developers:

- They allow you to build distributed applications by combining components developed using tools from different vendors.
- They make it easy to write applications. You do not have to deal with low-level details of transaction and state management, multithreading, resource pooling, and other complex low-level APIs. However, if necessary, expert programmers can still gain direct access to the low-level APIs.
- They can be developed once and then deployed on multiple platforms without recompilation or source code modification.
- The EJB specification that governs the use of enterprise beans is compatible with other Java APIs and CORBA. It also provides for interoperability between enterprise beans and non-Java applications.

Home interface
The *home interface* specifies the available methods for locating, creating, and removing instances of enterprise bean classes.

Remote interface
The *remote interface* specifies the remote business methods that a client can call on an enterprise bean.

EJB container
An *EJB container* is a run-time environment that manages one or more enterprise beans. The EJB container manages the life cycles of enterprise bean objects, coordinates distributed transactions, and implements object security. Generally, each EJB container is provided by an EJB server and contains a set of enterprise beans that run on the server.

Deployment descriptor
A *deployment descriptor* is a serialized object (javax.ejb.deployment.DeploymentDescriptor) that contains run-time settings for an enterprise bean and passes information to the EJB container about how to manage and control the enterprise bean.

Control descriptor
A *control descriptor* is a special class (javax.ejb.deployment.ControlDescriptor) that defines the database isolation levels and other constant values for each method in an enterprise bean that is used by the deployment descriptor.

EJB server
An *EJB server* is a high-level process or application that provides a run-time environment to support the execution of server applications that use enterprise beans. An EJB server provides a JNDI-accessible naming service, manages and coordinates the allocation of resources to client applications, provides access to system resources, and provides a transaction service. An EJB server could be provided by, for example, a database or application server.

Server configuration
A *server configuration* is one or more enterprise beans that, in combination with a set of specific properties, have been added to an EJB server.

RELATED CONCEPTS
Overview of the EJB Development Environment

Enterprise bean associated types

The following types are generated by the Create Enterprise Bean SmartGuide.

Type	*Name**	*Description*
Remote interface	MyBean	Specifies the remote business methods that a client can call on an enterprise bean.
EJB implementation	MyBeanBean	Provides the server-side implementation of the enterprise bean.
Home interface	MyBeanHome	Defines the methods used to create or find the enterprise bean.
Key class	MyBeanKey	Used as a method parameter to create or find an entity bean. It represents the “identity” of the entity bean. For example, it will represent the primary key columns of a row in a relational database.
Finder helper	MyBeanFinderHelper	Helper class for special finder methods for CMP entity beans.

* This table assumes that the name of the enterprise bean is *MyBean*. If you use another name, such as *TestBean*, the name of the generated types is modified as necessary; for example: TestBeanBean, TestBeanHome, etc.

The EJB menu item **Generate Deployed Code** also generates a number of different types, which are specific to the WebSphere Application Server EJB container.

RELATED CONCEPTS

Key classes and key fields

RELATED TASKS

Adding enterprise beans
Generating EJB deployed classes

Inheritance and association

In the EJB Development Environment, you can optionally define an inheritance relationship or an association relationship between different enterprise beans. The following sections describe the concepts of inheritance and association.

Note: Support for inheritance and associations has been included in VisualAge for Java as a technical preview. It can be used for deployment to the WebSphere Application Server, Advanced Edition, or for EJB JAR exportation to Component Broker (WebSphere Application Server, Enterprise Edition). Limitations exist for Component Broker deployment; see the Component Broker documentation for details.

Inheritance
The EJB Development Environment permits two forms of inheritance: standard class inheritance and EJB inheritance. In standard class inheritance, the home interface, remote interface, or enterprise bean class inherits properties and methods from base classes that are not themselves enterprise bean classes or interfaces. An exception is when you use the superclass field of the Create Enterprise Bean SmartGuide to specify a class that is the bean class of an enterprise bean. This allows you to have your enterprise bean classes or interfaces extend other enterprise bean classes or interfaces.

In EJB inheritance, by comparison, an enterprise bean inherits properties (such as CMP fields and association roles), methods, and method-level control descriptor attributes from another enterprise bean that resides in the same EJB group. To assist you in defining an EJB inheritance relationship, a SmartGuide is provided that allows you to create a new enterprise bean that inherits from an existing enterprise bean.

Some of the characteristics of EJB inheritance are:

- For CMP entity beans, there is support for single table and root-leaf table mapping. However, large hierarchies (wide or deep) might perform poorly because of the large size of the SQL statements. Root-leaf mapping is more susceptible to this issue because of the number of joins involved in the SQL statement.
- The deployment descriptor will list all of the CMP fields for each enterprise bean, including those fields that have been inherited.
- Home interface classes cannot inherit from other home interface classes. When an inheritance relationship is defined between enterprise beans, all home

methods that are not part of an association are copied to the child bean. The child home interface will still only extend javax.ejb.EJBHome.

- The bean class of the new child enterprise bean will extend the bean class of the parent enterprise bean. No parent bean class methods will be defined in the child bean class.
- Remote interface classes can inherit from other remote interface classes.
- Key classes are common to all inherited enterprise beans. This means that the key class in the child enterprise bean is identical to the key class in the parent enterprise bean.

Information about creating a new enterprise bean that inherits from an existing enterprise bean is found in the topic "Adding enterprise beans with inheritance".

Association

In the EJB Development Environment, an association is a relationship that exists between two CMP entity beans within an EJB group. There are three main types of association: one-to-one, one-to-many, and many-to-many.

In a one-to-one (1:1) association, a CMP entity bean is associated with a single instance of another CMP entity bean. For example, an Employee bean could be associated with only a single instance of an EmployeeID bean, because an employee can have only one employee identifier on file.

In a one-to-many (1:M) association, a CMP entity bean is associated with multiple instances of another CMP entity bean. For example, a Department bean could be associated with multiple instances of an Employee bean, because most departments are made up of multiple employees.

In a many-to-many (M:M) association, multiple instances of a CMP entity bean are associated with multiple instances of another CMP entity bean. A many-to-many association is created by joining two 1:M associations. For example, multiple instances of a Customer bean could be associated with multiple instances of a Restaurant bean, because restaurants serve many customers, and customers patronize many different restaurants.

Use the Association Editor to specify the association between two enterprise beans in your group. The Association Editor generates the appropriate finder methods to support any associations that you create.

As part of defining an association, you assign a *role* to each bean relative to the other bean, and you give that role a name. For example, suppose you create an association between Employee and EmployeeID. The role of EmployeeID within the Employee bean could be something meaningful like *id* or *employeeID*. The role of Employee within the EmployeeID bean could be something like *employee* or *owner*. These names are used to derive method names in the generated code and become part of the enterprise bean's remote interface.

To define cardinality for the association, you assign multiplicity to each role. For more information about defining cardinality, see F1 help for the Association Editor.

In the Enterprise Beans pane of the EJB page, icons indicate the presence of associations and cardinality.

- If you select an EJB group that contains associated enterprise beans, the names of all of the associations appear in the Properties pane, and the following icon appears to the left of each association: 

- If you select an individual enterprise bean that is associated with another enterprise bean, the names of all roles appear in the Properties pane beside their corresponding CMP fields, and the following icon appears to the left of each role: 
- Icons appear to the right of the roles to indicate whether the relationship is one-to-one, one-to-many, or many-to-many. For example, the following side-by-side icons indicate a one-to-many relationship: 

When CMP fields and associations are eventually mapped to database tables, foreign keys are used to represent the associations in the database tables. You can then add association roles to the key of an enterprise bean, which makes the foreign key represented in the association part of the key class.

Preserving inheritance and association information when exporting and importing
If you want to export and later import enterprise beans involved in an inheritance or association relationship (for example, to upgrade to a more recent version of VisualAge for Java), you must export the project containing the beans into a repository (.dat) file to preserve the information that defines the inheritance or association. When you later import the .dat file, the inheritance and association information is preserved. Inheritance and association information is not preserved if you import the enterprise beans from an EJB JAR file.

RELATED CONCEPTS

Key classes and key fields

RELATED TASKS

Adding enterprise beans with inheritance
Creating or editing associations

Approaches for mapping enterprise beans to database tables

The EJB Development Environment enables the following development approaches for mapping enterprise beans to database tables:

- Top-down
- Bottom-up
- Meet-in-the-middle

Top-down
This approach assumes that you already have existing enterprise beans contained in an EJB group. In this approach, the enterprise bean design determines the database design.

After you have finished defining your enterprise beans within an EJB group, you can generate a schema and map from the EJB group.

The generated schema contains one table for each entity bean with container-managed persistence (CMP) that belongs to the selected EJB group. In these tables, each column corresponds to a CMP field of the enterprise bean, and the generated mapping maps the field to the column. Associations are mapped to foreign-key relationships.

By default, EJB inheritance hierarchies are mapped to a single table; that is, the base and all derived enterprise beans are mapped to the same database table. However, if a root-leaf map exists for the root of the inheritance hierarchy and new enterprise beans are added to the hierarchy, the new beans will also be mapped root-leaf.

After the default schema is created, you export the schema to the database. For more information, see the topic "Generating a schema and mapping from an EJB group."

Bottom-up
This approach assumes that the database tables already exist.

In this approach, you import the schema from the database into the EJB Development Environment using the Schema Browser. After you have imported the schema, you can generate the enterprise beans, an EJB group, and the mappings between them.

Information about creating EJB groups from existing database schemas or Persistence Builder models is found in the topic "Creating EJB groups from schemas or models."

Meet-in-the-middle
This approach assumes that you are going to create your enterprise beans and database schema simultaneously but independently. You can also use this approach to modify existing enterprise beans that have been created using the top-down or bottom-up approaches.

To create your schema, you can import the tables from a database using the Schema Browser. After you have populated an EJB group with enterprise beans and created the schema, you can create a new EJB group map using the Map Browser.

In this approach, you map each field of the enterprise bean to the corresponding column of a table within the selected schema. You also map each association role to a foreign-key relationship.

Repeat these steps until all components from the EJB group are mapped to a component within the schema. For more information about the individual steps in this process, see topics in the Related Tasks list below.

RELATED TASKS

Generating a schema and mapping from an EJB group
Creating EJB groups from schemas or models
Creating schemas
Adding enterprise beans
Adding EJB groups
Creating EJB group maps

Converters

At times, you might need to convert data being read from a database or saved to a database. *Converters* translate a database representation to an object type, and back.

For example, you might want to convert a CHAR database entry 'Y' to a Boolean object set to `true`. In this case, specify the VapStringToBooleanConverter when you create the column for the data in the Schema Browser.

For many column types, you can use the default converter, VapConverter, because the object type can be derived satisfactorily from the database representation. VapTrimStringConverter is handy for strings; it strips off leading and trailing blanks from the data being read.

In most cases, the name of the converter specifies the database and object types. The first type specified is that of the database.

For a list of converters shipped with VisualAge for Java, see the topic "Data-type converters supported for persistence."

If you wish to deploy your CMP beans to Component Broker, be aware that usage of VisualAge for Java converters is not directly supported there. However, the Object Builder tool can automatically handle the mapping of some data types in preparation for deployment to Component Broker. For more information, see "Default type mappings for deployment to Component Broker."

RELATED TASKS

Writing your own converters
Creating tables and columns

RELATED REFERENCES

Data-type converters supported for persistence
Default type mappings for deployment to Component Broker

Key classes and key fields

A *key class* is used to create or find an entity bean. It represents the identity of the entity bean, corresponding to the primary-key columns of a row in a relational database.

Each designated container-managed field in the entity bean that corresponds to one of these columns is called a *key field*. Each key field is a member of the entity bean's key class. The following icon appears next to the name of each key field:

As an example, suppose you have a VapAccount bean for a banking application. The accountNumber property has been designated as a key field; it corresponds to the primary key of an ACCOUNT database table. The key class for VapAccount (VapAccountKey) includes the following:

- A public String field called accountNumber
- A one-argument constructor, VapAccountKey(String), that sets accountNumber

Key classes are common to all inherited enterprise beans. This means that the key class of a parent enterprise bean is also used by all of its child enterprise beans. Key classes can be shared only among beans in the same inheritance hierarchy.

Key fields added through association
Key fields are duplicated in beans that hold foreign keys for associations. When such a bean becomes part of an association, the constituent key fields of the other

bean are automatically copied into the first bean's bean class. The name of the new field follows the convention *roleInThisBean_keyFieldFromOtherBean*. The type of the new field remains the same as in the original, except that primitive types are converted to object types (for example, int to java.lang.Integer).

For example, suppose a 1:M association between VapAccount and VapCustomer enterprise beans (one customer can have at least one account). Each instance of VapAccount holds a foreign key (customerNumber) to its associated VapCustomer instance. When the association is defined, the following field is added to VapAccount:

```
public java.lang.String customer_customerNumber;
```

You can also choose to add an association role to the key of an enterprise bean. This copies the foreign key represented in the association into the key class of the enterprise bean as well as into its bean class.

RELATED TASKS

Defining CMP fields and key fields
Adding or removing association roles in keys

Access beans: overview

The Sun EJB specification describes a server-side component architecture that features the use of enterprise beans. Two interfaces define the way that a client program accesses these enterprise beans:

- Home interface
- Remote (EJBObject) interface

The home interface contains methods that describe how you can instantiate an enterprise bean object. The remote interface, by comparison, defines the methods of an enterprise bean that can be accessed by your user program. To access an enterprise bean, your user program goes through the following steps:

1. Obtains a context to the name server (name service context).
2. Looks up the home of the enterprise bean using the name service context.
3. Creates an enterprise bean instance from the enterprise bean home, which returns an enterprise bean proxy object.
4. Accesses the remote methods of the enterprise bean instance through an enterprise bean proxy object. Each call to the enterprise bean proxy object is a remote call that can throw an exception, such as java.rmi.RemoteException.

When you program directly to the enterprise bean interfaces, you increase the complexity of your user program and can incur significant performance problems. Each call to the enterprise proxy object is a remote call, so accessing a large number of entity bean attributes can take a significant amount of time. These problems, however, are now largely solved by the use of *access beans*.

In the EJB Development Environment, access beans are Java bean wrappers for enterprise beans that are typically used by client programs, such as Java ServerPages (JSP) files, servlets, and sometimes even other enterprise beans. Access beans adapt enterprise beans to the JavaBeans programming model and hide the complexity associated with the home and remote interfaces. They provide fast access to enterprise beans by letting you maintain a local cache of enterprise bean attributes. Access beans make it possible to use an enterprise bean in much the same way that you would use a Java bean. (Access bean creation tools are

provided only in the EJB Development Environment. No support for access bean creation tools exists outside of VisualAge for Java.)

VisualAge for Java provides a SmartGuide to assist you in creating or editing access beans. The SmartGuide automatically saves your property settings in the repository, so that when you later generate access beans, you do not need to specify the settings again. Information about creating or editing access beans is found in the topic "Creating or editing access beans."

Types of access beans
There are three types of access beans (listed in order of increasing complexity):

- Java bean wrapper (for a session or entity bean)
- Copy helper (for an entity bean)
- Rowset (for multiple entity bean instances)

The SmartGuide shields you from any differences in complexity between the three types of access beans. When you use the SmartGuide, a rowset is as easy to create as a simple Java bean wrapper.

Once you have created your access beans, whenever you create a client JAR file for your enterprise beans, all of the access beans that exist for the selected enterprise beans will be exported to the JAR file. Each access bean class will be marked as a Java bean in the manifest file.

The three different types of access beans are discussed in greater detail in the following three topics:

- "Access beans: Java bean wrappers"
- "Access beans: copy helpers"
- "Access beans: rowsets"

General characteristics of access beans
Access beans have the following general characteristics:

- Home interface methods that return a single instance of EJBObject are mapped to Java bean constructors, while remote interface methods are mapped to Java bean methods. Each finder method in the home interface that returns a collection of enterprise bean instances is mapped to a finder method in the access bean. You must first instantiate the access bean and then invoke the appropriate finder method that will return a collection of access bean instances.
- You can select the enterprise bean for which you want to create an access bean, then you can use a SmartGuide to customize and create the access bean. For example, the SmartGuide allows you to choose the home interface method that you want to use to map to the no-arg access bean constructor. The arguments, however, must be set by special setter methods and stored as instance variables in the access bean.
- To instantiate an enterprise bean, the access bean invokes a create() or finder method defined in the enterprise bean home interface. If a no-arg constructor is used, the access bean only instantiates the actual enterprise bean when the first business method is called.
- Access beans employ copy helper objects that are basically caches of user-selected entity bean attributes that are stored inside the access bean. The getter and setter methods for these attributes deal directly with the local cache rather than calling straight through to the remote getter and setter call. Methods are provided to flush the cache to the actual enterprise bean database and to refresh the cache from the actual enterprise bean. This can improve performance

significantly for entity enterprise beans that have a large number of attributes, where issuing one remote call to get and set a large number of attributes is faster than issuing a single remote call for each.

- To facilitate the use of access beans in constructing JSP pages that are text-oriented, a String conversion feature is provided for the getter and setter methods of user-selected attributes, as well as the setter methods for the arguments of the home interface method (chosen to map to the no-arg constructor). You can employ the provided SimpleStringConverter for Java primitive types and their Object wrappers or provide your own converter for more complex types.

General access bean conventions
Access beans follow these conventions:

- If you are creating converter methods in your own StringConverter class, you must adhere to the following conventions:

  – The converter class must implement the com.ibm.ivj.ejb.runtime.StringConverter interface.
  – For each type *XXX* that you want to support, you must provide the methods StringTo*XXX*(String arg) and *XXX*ToString(*XXX* arg). Also, for types that are arrays, the required methods are StringTo*XXX*Array*Y*(String arg) and *XXX*Array*Y*ToString(*XXX* arg), where *Y* is the dimension of the array. (This is only required if the dimension is two or higher.)

    Examples:

    For a Java type com.acme.FooBar, the required methods are StringToFooBar(String arg) and FooBarToString(FooBar arg).

    For a Java type int[][], the required methods are StringTointArray2(String arg) and intArray2ToString(int[][] arg).
  – You can use the SimpleStringConverter for the Java primitive types, such as Integer and Float.

- Ensure that you are following the JavaBeans conventions for defining getter and setter methods for your enterprise bean properties. If you don't have getter and setter methods, then you won't be able to add the corresponding fields to the copy helper. Note that the getter and setter methods also should not throw any exceptions because the JavaBeans rules for getter and setter methods do not allow for this.

In addition to the information about access beans in this topic and in the topics listed below in the Related Concepts and Related Tasks, you can find more information about access beans in the paper entitled "Developing EJB Access Beans in VisualAge for Java", which is available from the VisualAge Developer Domain (VADD) Web site at the following URL:

www.ibm.com/software/vadd

RELATED CONCEPTS

Access beans: Java bean wrappers
Access beans: copy helpers
Access beans: rowsets

RELATED TASKS

Creating or editing access beans
Creating client applications using access beans

Access beans: Java bean wrappers

Of the three types of access bean, a Java bean wrapper is the simplest. It is designed to allow either a session or entity enterprise bean to be used like a standard Java bean and it hides the enterprise bean home and remote interfaces from you. Each Java bean wrapper that you create extends the following superclass:

com.ibm.ivj.ejb.access.AccessBean

A Java bean wrapper access bean has the following characteristics:

- It contains a no-arg constructor.
- When the SmartGuide prompts you to map one of the create() or finder methods defined in the home interface to the no-arg constructor of the access bean, the access bean will subsequently contain one init_*xx* property for each parameter of the create() or finder method that was mapped to the no-arg constructor. To simplify a JSP program that normally handles the String type, you can choose to have your access beans expose the init_*xx* properties as String types. However, you can also select your own converters for the init_*xx* properties.
- When a key class is used in the create() and finder methods for a CMP entity bean, the key fields are used as the init_*xx* properties instead of the key class. A key field is normally declared as a simple type. This makes it easier for visual construction tools, such as the Visual Composition Editor, to use an access bean.
- When the no-arg constructor is used, the init_*xx* properties must be set first before any other calls to the access bean.
- The access bean may contain several multiple-arg constructors, each corresponding to one of the create() or finder methods defined in the enterprise bean home interface.
- The access bean run time contains a class for handling enumerations of EJBObjects. For any home finder method that returns a java.util.Enumeration of EJBObjects, the corresponding access bean method now returns a specialized Enumeration class whose nextElement() method instantiates the enterprise bean on the server side and returns an access bean instantiated using the corresponding EJBObject reference. The enterprise bean is only instantiated when nextElement() is called rather than when the finder method is called on the access bean. Also, you do not need to instantiate the access bean manually (based on the EJBObject reference).
- The access bean will perform lazy initialization when the no-arg constructor is used. When the access bean is instantiated, it will *not* instantiate the enterprise bean. On a remote method call, the access bean will first instantiate the remote enterprise bean if it has not yet been instantiated.
- A default JNDI name will be generated into each access bean class. The code generator will read the deployment descriptor and pass the JNDI name to the access bean. You can change the JNDI name using the setInit_JNDIName() method. It is not expected that you will need to change the JNDI name. However, in the event that an enterprise bean is deployed into a different home, the administrator may add a prefix to the JNDI name to indicate the different home.
- To look up a home, an access bean needs to obtain a name service context, which is sometimes known as the rootContext. A rootContext can be constructed

if you know the name service URL and the name service type. Access beans provide two APIs to set the properties of the context:

setInit_NameServiceTypeName()
setInit_NameServiceURLName()

However, if a client program is running in the WebSphere run-time environment, you do not need to use these two APIs because the rootContext is set automatically.

For setting the global JNDI URL name and type for all access beans, two pairs of APIs are provided. The first pair of APIs is:

- public void setInit_NameServiceTypeName(String name)
- public void setInit_NameServiceURLName(String name)

The above methods are instance methods, which will modify the name service properties of the access bean they are called on. They will only be used if the no-arg constructor of the access bean is used.

The second pair of APIs is:

- public static final setGlobal_NameServiceTypeName(String name)
- public static final setGlobal_NameServiceURLName(String name)

These methods are static methods and will apply to all access beans used in the client program. Before instantiating a new access bean, call these methods in order for the particular access bean to use the new name service type or URL when it instantiates the enterprise bean on the server side.

- An enterprise bean remote interface method can return an enterprise bean object. When this kind of method is generated in the access bean class, the return type is changed to the corresponding access bean type. This allows your user program to deal with only the access bean type and inherit the benefits provided by the access bean.
- When multiple instances of an access bean use the same home (for example, that use the same JNDI name and rootContext), the access bean class only looks up the corresponding enterprise bean home once. Each access bean class retains some class-level cache to improve the performance when instantiating an enterprise bean. Note that during the lifetime of your client's VM, the access bean run time keeps a static cache of all home references indexed by the JNDI name, name service URL, and name service type. The client can issue a call to resetHomeCache() to reset the cache. (This is useful in the event that a particular home reference is no longer valid.)
- When access beans are serialized, the following access bean properties do not get serialized:

  - remote interface references (EJBObject references)
  - home interface references
  - global JNDI information (name, name service URL, name service type)

  (Note that for copy helper and rowset access beans, the cache of attributes is serialized.)

In the following example, the user program to create an employee is shown:

```
EmployeeAccessBean aEmployee = new EmployeeAccessBean ()
aEmployee.setInit_employeeNo ("100");
aEmployee.setName ("IBM");
aEmployee.setAddress ("1150 Eglinton Ave, Toronto");
```

RELATED CONCEPTS

Access beans: overview
Access beans: copy helpers
Access beans: rowsets
Key classes and key fields

RELATED TASKS

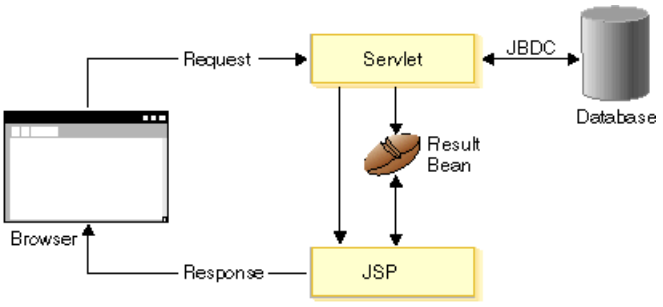
Creating or editing access beans
Creating client applications using access beans

Access beans: copy helpers

A copy helper access bean has all of the characteristics of a Java bean wrapper access bean. These characteristics are described in the topic "Access beans: Java bean wrappers", which you should ensure that you are thoroughly familiar with prior to working with copy helper access beans. Helpful information is also found in the topic "Access beans: overview".

Unlike a Java bean wrapper access bean, a copy helper access bean also incorporates a single copy helper object that contains a local copy of attributes from a remote entity bean. A client program will retrieve entity bean attributes from the access bean object, as it would for a Java bean wrapper. If those attributes are maintained in the copy helper object, the access bean will retrieve them from the copy helper object without making a remote call.

An entity bean can contain a large number of attributes. In most situations, these attributes are used to create an HTML output page by a JSP program. As illustrated in the following figure, a servlet typically invokes a command (such as a request to an entity bean) and then invokes the JSP program to display the result.



Depending on the results returned by the command, the servlet may invoke a different JSP program. For the associated results bean, there are a couple of expectations:

- A call to retrieve an attribute should be fast because there can be many attributes that a JSP program needs to retrieve.
- A call to retrieve an attribute should not cause an exception because it will needlessly complicate the JSP logic.

A Java bean wrapper access bean does not meet the requirement for the JSP programming model, because every call is a remote call that can throw a remote exception and that has a long execution call path. However, you can use a copy helper access bean to resolve these concerns.

When you create a copy helper access bean, the remote interface of the enterprise bean will be changed to extend the CopyHelper interface as well as the EJBObject interface. You can select all of these attributes or only a subset in creating the copy helper object. The selected attributes are saved in the enterprise bean meta model. These selections are redisplayed if you decide you want to change the selection.

The copy helper object is stored inside the access bean. A get() and set() method is delegated to the local copy helper object instead of the remote enterprise bean object. To commit the changes in the copy helper to the remote enterprise bean or to refresh the local copy helper from the remote enterprise bean, your user program must call the commitCopyHelper() method and the refreshCopyHelper() method, respectively.

When you create a copy helper access bean, a copy helper interface is added to the corresponding EJBObject interface. There are two methods defined in the copy helper interface:

- _copyToEJB()
- _copyFromEJB()

You do not need to implement these methods. They are automatically generated into the bean class when you generate the access bean.

An AccessBeanData interface class containing abstract representations of the getter and setter methods is also generated. The actual access bean class implements this interface. This can serve as a data object to be used in a JSP which is independent of the access bean implementation.

Along with user-selected enterprise bean attributes, the key object is also stored in the copy helper cache as soon as the enterprise bean is instantiated by the access bean. To get the key object from an access bean, use the __getKey() method.

When you call a getter method for a cached attribute, the access bean follows the protocol below:

1. If the cache is empty or only contains the key object, then a complete cache refresh is performed using refreshCopyHelper().
2. If the cache does not contain the key object, it is first flushed to the enterprise bean (if it is not empty) to update any attributes that might have been set, then it is completely refreshed.
3. Finally, the particular attribute's value is retrieved from the cache. (For more information, see the information on IBM APIs in the Reference online help for the EJB Development Environment.)

A JSP program normally expects the String type to construct the HTML page. VisualAge for Java provides a SimpleStringConverter for simple types and you can also create your own String converter and specify it in the Create Access Bean Smart Guide.

The following example shows a user program that is used to create an Employee enterprise bean. (You need to know when to call the commitCopyHelper() and refreshCopyHelper() methods.)

```
EmployeeAccessBean
aEmployee = new EmployeeAccessBean ()
aEmployee.setInit_employeeNo (100);
aEmployee.setName ("IBM");
aEmployee.setAddress ("1150 Eglinton Ave, Toronto");
aEmployee.commitCopyHelper() <-- instantiate remote bean,
                             <-- commit changes to bean
```

The following example shows a user program that accesses an Employee enterprise bean.

```
EmployeeAccessBean
aEmployee = new EmployeeAccessBean ()
aEmployee.setInit_employeeNo (100);
aEmployee.setInit_departmentNo (1000);
String name = aEmployee.getName();
String address = aEmployee.getAddress();
```

For an extended example of how to use copy helper and rowset access beans, see the topic “Example: Using copy helper and rowset access beans”.

RELATED CONCEPTS

Access beans: overview
Access beans: Java bean wrappers
Access beans: rowsets

RELATED TASKS

Creating or editing access beans
Creating client applications using access beans

RELATED REFERENCES

Example: Using copy helper and rowset access beans

Access beans: rowsets

A rowset access bean has all of the characteristics of both the Java bean wrapper and the copy helper access beans. These characteristics are described in the topics “Access beans: Java bean wrappers” and “Access beans: copy helpers”, which you should ensure that you are thoroughly familiar with prior to working with rowset access beans. Helpful information is also found in the topic “Access beans: overview”.

Unlike a copy helper access bean that contains a single copy helper object, a rowset access bean contains multiple copy helper objects. Each copy helper object corresponds to a single enterprise bean instance.

In some cases, a JSP program needs to construct a table from a collection of entity beans. The rowset access bean allows you to build a collection of entity bean instances. You can view a rowset access bean as a table, with each row corresponding to the attributes of an enterprise bean instance. The access bean exposes each row as indexed properties to support JSP programs. (Currently, JSP programs cannot handle the Enumeration type.)

Rowset access beans, as well as Java bean wrapper and copy helper access beans, are serializable. For this reason, session bean methods should use rowset objects as a return type. For example, a session bean performing a search function will return the result set as a rowset object.

A rowset access bean contains a collection of copy helper objects. In turn, a copy helper object contains the primary key for each entity bean instance, but it does not contain the proxy object (the EJBObject in the EJB server) itself for the entity bean. When a session bean returns a rowset access bean as a result set, only the attributes of the entity beans are copied to the client space. The proxy objects are not copied. This is because copying a large number of enterprise bean proxy objects from the server space to the client space can cause performance problems. A JSP program can read from a rowset access bean immediately without invoking a remote call. On an update call, such as might be made using the commitCopyHelper() method, the access bean constructs the enterprise bean proxy object using the key object saved in the copy helper.

In the following example, the user program is shown:

```
Test.EmployeeAccessBeanTable
es = new Test.EmployeeAccessBeanTable();
Test.EmployeeAccessBean e = null;
Test.SearchAccessBean s = new Test.SearchAccessBean ();
es.setEmployeeAccessBean(findEmployeesInDepartmentWithSalary (10000,"Payroll"));
try {
   for (int i=0;;i++;) {
      e = es.getEmployeeAccessBean(i);
      e.getName(); // read current row
      e.getSalary();
   }
}

catch (IndexOutOfBoundsException o) {}

try {
   for (int i=0; ; i++) {
      e = es.getEmployeeAccessBean(i);
      e.setSalary (es.getSalary() * 1.15); // 15% increase for everybody
      e.copyToEJB ();
   }

catch (IndexOutOfBoundsException o) {}
```

For an extended example of how to use copy helper and rowset access beans, see the topic “Example: Using copy helper and rowset access beans”.

RELATED CONCEPTS

Access beans: overview
Access beans: Java bean wrappers
Access beans: copy helpers

RELATED TASKS

Creating or editing access beans
Creating client applications using access beans

RELATED REFERENCES

Example: Using copy helper and rowset access beans

Relationship between the EJB page and the reserved package

The EJB page in the Workbench is where you perform all of your EJB development activities. This allows you to work directly with EJB groups and enterprise beans, and it gives you access to the operations supported by the EJB Development Environment.

There are several ways you can add EJB groups and enterprise beans to the EJB page:

- Create new EJB groups (from scratch, or from schemas or Persistence Builder models) and create new enterprise beans.
- Retrieve existing EJB groups and enterprise beans from the repository.
- Import existing EJB groups or enterprise beans from a JAR file.

In the EJB Development Environment, your enterprise beans reside in *EJB groups*, which are logical groups in which you can organize your enterprise beans. These EJB groups also mark the boundary for those enterprise beans that participate in inheritance or association relationships with each other. For each EJB group, the meta data that encapsulates the organizational information for the group is held in a special package known as the *reserved package*. Although this package is visible in the IDE, the meta data it contains is visible only to the EJB Development Environment. As a result, you must always use the EJB page of the Workbench to manipulate enterprise bean entities. For easy identification, the name of a reserved package takes the following form:

*EJB_group_name*EJBReserved

Instead of a package to hold the meta data associated with an enterprise bean, a Java class element is used. These Java class elements are placed in the reserved packages of their corresponding EJB groups. Although the classes are visible in the IDE, you should never attempt to directly delete, replace, reorganize, or otherwise manage these classes. In addition, "normal" Java classes for the enterprise beans should never be placed in the reserved package.

Because a reserved package is intended solely as a container for the meta data associated with an EJB group, you should not put any enterprise-bean code in a reserved package. However, it is reasonable to use the reserved package to hold other dummy Java classes that contain meta data. An example would be the database schema and map classes associated with an EJB group. Putting the meta data-filled schema and map classes in the reserved package serves as a reminder that each version of EJB group is a one-to-one correspondence to the schema and map classes for that group. Actual meta data version synchronization between the EJB group and the associated schema and map classes is still primarily a user responsibility, but when carefully done at version time, you will not need to worry if loading a version of an EJB group also requires the loading of another package for the corresponding schema and map meta data. In this way, the reserved package can, by itself, be used as a concise unit of meta data.

RELATED CONCEPTS

Editions and versioning
EJB development in a team environment

RELATED TASKS

Adding EJB groups
Adding enterprise beans

Editions and versioning

Working with editions of EJB groups and enterprise beans in the EJB Development Environment is somewhat different than working with other elements in the IDE. Before examining the detailed behavior involved in creating open editions and in versioning EJB groups and enterprise beans, it is important to remember that each EJB group is represented by a reserved package and each enterprise bean is represented by a corresponding Java class in the reserved package. Additional information about EJB groups and reserved packages is found in the topic "Relationship between the EJB page and the reserved package".

As a result, when you are working with EJB groups and enterprise beans in the EJB page, you are affected by the IDE team environment rules that are applicable to those representative Java classes contained in the reserved package of the project. It is important to understand the authority-related implications imposed by the IDE team environment. Information about these authority-related implications is found below in the section "Implications of IDE authorization rules".

Open editions
When you need to make changes to an EJB group or enterprise bean, the group or object must be in the open edition state. Note that the source code package must be an open edition, which you can create in the Projects page. However, you should never create an open edition of the reserved package from the Projects page. It will be done automatically when you create an open edition of an EJB group in the EJB page. In the open edition state, you can change the contents of enterprise beans or their associated generated Java classes and interfaces. Similarly, EJB groups must be in the open edition state before you can add new enterprise beans. (You must add, delete, or replace an enterprise bean in an EJB group from within the EJB page.)

A new open edition is automatically created for you whenever you try to modify a versioned enterprise bean. However, only a scratch edition is created for the reserved package and source code package. You must be a group member of the reserved and source packages if you are creating a new enterprise bean or if you are adding associations.

Versioning
In the EJB Development Environment, versions allow you to track the design evolution and relationships between the different EJB page elements, such as EJB groups, enterprise beans, and the Java classes and interfaces generated from the enterprise beans. Versioning an EJB element on the EJB page will result in a versioned representative Java class in the reserved package for the EJB element. All generated Java classes associated with the enterprise bean will also be versioned if they are not already.

Versioning an EJB group will version the corresponding reserved package, which in turn causes the EJB elements, as well as enterprise bean source code and generated classes, to be versioned. However, you will need to manually version the enterprise bean source code packages and projects, as needed. Versioned classes can always be retrieved from the repository when an EJB version is loaded.

Note that the database schema and map classes are not automatically versioned when operations are performed on the EJB page. If you choose to save your schema and map classes in the reserved package of the corresponding EJB group, you must first version your schema and map classes before you version the EJB group. Otherwise, the EJB group version operation will fail. Saving your schema

and map classes in the reserved package will allow later retrieval from the repository without needing to load the schema and map classes separately. Furthermore, if the schema and map classes were saved properly, you don't need to be concerned that the version of the schema and map classes will not be synchronized with the EJB group version.

When you load a versioned project that contains enterprise beans, you can view those enterprise beans in the EJB page when the page is opened. However, if you want to add or delete enterprise beans or associations, you must first go to the Projects page in the IDE and create a new edition of the bean packages contained in the project, much like you would any other package. Since EJB groups are related to reserved packages, and enterprise beans are related to Java classes in the reserved package, you must know the implications of the IDE authorization rules, which are discussed in the following section.

Implications of IDE authorization rules
Since the classes in the reserved package hold meta data for the EJB groups and enterprise beans, the version states of the EJB groups and enterprise beans are directly reflected in the representative classes. Additionally, because these representative classes must adhere to the IDE team environment rules, the versioning of EJB groups and enterprise beans is affected by the rules as well. For example, if you are not defined as a member of the reserved package, you will not be allowed to create any class in the reserved package. As a result, you will not be able to create enterprise beans in the EJB page for that group.

To reduce the burden of administering user access for EJB developers, it is recommended that the developer who works with the EJB Development Environment also serve as the owner of the associated project. This will help ensure that the EJB developer will not encounter authority problems when creating additional enterprise beans, open editions of EJB groups and enterprise beans, or new packages in the project. Similarly, since the compilation of enterprise beans produces Java types that are imported into the IDE, the developer responsible for generating the Java types must either be the owner or the recognized developer of the Java package into which the generated Java types are saved.

If the Java package that is to hold the generated Java types does not yet exist in either the repository or the workspace, and if the EJB developer owns the project in which the package will reside, then there should be no problems with authorization when generating Java classes and creating the new packages. This is because when you own a project, you have the authorization to create and own Java packages contained in the project, as well as all generated Java types.

Schemas and maps
In VisualAge for Java, to retain any changes you make to schemas and maps, you must explicitly save the schemas and map as Java classes. Otherwise, your changes will be lost. You must also save your schemas and maps to share them with other developers. When you save your schemas and maps, you must provide a project and package to save them in. Saving a schema or map puts the current state of the meta data in the specified Java storage class.

Although it is not a requirement for you to save your schemas and maps in the same project that your bean code is in, we recommend that you do.

To save your schemas and maps, use the **Schemas - Save Schema** and **Datastore_Maps - Save Datastore Map** menu items, available from the Schema and Map browsers, respectively.

The classes created by saving schemas and maps do not get versioned when you version the EJB groups. You need to version them yourself. For this reason, you may want to ensure that all of your code resides in the same project, so that you can just version the entire project at once rather than versioning individual elements of the project. If you are saving your schema and map classes in the reserved package, you must version them before versioning the EJB group representative of the reserved package.

If you have previously generated and deleted a schema and map, be sure to delete the corresponding storage classes before regenerating.

RELATED CONCEPTS

EJB architecture
Relationship between the EJB page and the reserved package
EJB development in a team environment

RELATED TASKS

Versioning EJB groups and enterprise beans
Creating open editions of EJB groups and enterprise beans
Defining database schemas: overview
Creating schemas
Mapping database schemas: overview

EJB development in a team environment

In a team environment, when enterprise beans need to be shared among multiple developers, you should create a version of the project that contains your enterprise beans and share the beans at the project level. The reason is that the EJB meta model, schema, and map can only be loaded in another workspace with a version of the containing project, not packages. Sharing beans at the project level can also help you avoid authorization problems when individual Java packages are owned by different users.

Whenever an EJB group is loaded, the map and schema are searched and loaded automatically if they are found. If you load a version of the reserved package, the currently loaded version of the map and schema will be loaded. This is why loading at the project level is recommended, so that the correct version of the map and schema will be loaded. If the map and schema are saved in the reserved package, you should always get the correct version of the map and schema.

In a team environment where enterprise beans can be shared, the project owner is responsible for merging all of the enterprise beans and all of the corresponding schema classes and map classes, then creating an edition of the project so that the team members can share the project. An individual team member cannot replace an edition of an enterprise bean without proper authority to the bean class and all corresponding classes.

In general, it is recommended that you use a local repository for EJB development. EJB code generation can take an unnecessarily long time when you are using a remote repository. However, a remote repository is recommended for sharing versioned code.

Although you can use the VisualAge for Java team development tools to develop your enterprise beans, it tends to present more challenges than developing other software components in a VisualAge for Java team environment. As a general rule,

we recommend that you share a single user ID for team development of enterprise beans. Also, you should generally have no more than one developer working on any given EJB group. However, if you would like to explore other alternatives for working with enterprise beans in a team development environment, see the "best practices" paper entitled "EJB Team Development in VisualAge for Java", which is available from the VisualAge Developer Domain (VADD) Web site at the following URL:

http://www.ibm.com/software/vadd

RELATED CONCEPTS

EJB architecture
Relationship between the EJB page and the reserved package
Editions and versioning

RELATED TASKS

Versioning EJB groups and enterprise beans
Creating open editions of EJB groups and enterprise beans
Defining database schemas: overview
Creating schemas
Mapping database schemas: overview

Limitations

This topic contains information about limitations for the EJB Development Environment. Additional information about current limitations, considerations, and known problems is found in the Release Notes for the EJB Development Environment.

The following limitations and restrictions exist for the EJB Development Environment.

- Support for inheritance and associations has been included in VisualAge for Java as a technical preview. It can be used for deployment to the WebSphere Application Server, Advanced Edition, or for EJB JAR exportation to Component Broker (WebSphere Application Server, Enterprise Edition). Limitations exist for Component Broker deployment; see the Component Broker documentation for details.
- You cannot associate an enterprise bean that resides in one group with an enterprise bean that resides in a different group. Associated enterprise beans must reside in the same group.
- It is difficult to move enterprise beans without risking errors, such as moving enterprise beans from one group to another.
- Enterprise beans cannot share the same single key class unless they are involved in an inheritance relationship.
- In a team environment, EJB groups can only be loaded into another workspace at the project level. The containing project must be versioned.
- In a team environment, a maximum of one developer should own an EJB group and its components.
- You cannot run security-related methods in the javax.ejb.EJBContext interface while running in the IDE. Security function is turned off in the simplified EJB Development Environment and calls to those functions will fail. If you need to test the security functions, you must test them by deploying the code to a production server, such as the WebSphere Application Server.

- You must use the EJB page to manage your enterprise bean's reserved package and its contents, and you should never attempt to directly delete, replace, reorganize, or otherwise manage the classes contained in the reserved EJB package of the project.
- After an enterprise bean is created, you cannot rename the bean classes and interfaces. To work around this limitation, delete the enterprise bean without deleting the bean classes and interfaces. (Information about deleting enterprise beans and associated components is found in the topic "Deleting EJB groups and enterprise beans".) Rename the bean classes and interfaces from an IDE page other than the EJB page, and then recreate the enterprise bean using the renamed bean classes and interfaces.
- You cannot reparent an enterprise bean once it has been created. You must delete the class and then recreate it.

RELATED TASKS

Deleting EJB groups and enterprise beans
Handling problems caused by DB2 connection or application limits

Chapter 2. Using and setting up the EJB Development Environment

Using the EJB Development Environment: overview

Once you have set up the EJB Development Environment by loading the required features and adding a JDBC driver to the class path, you can use the various tools of the EJB Development Environment to develop and deploy enterprise beans for use in Enterprise applications. (Information about loading the required features is found in the topic "Loading the required features". Information about adding a JDBC driver to the class path is found in the topic "Adding a JDBC driver to the class path".)

Typically, the development steps you would follow when using the EJB Development Environment are:

1. Add an EJB group to hold your enterprise beans.
2. Add enterprise beans, or import enterprise beans, to your EJB group.
3. Add the home methods and add the remote (business) methods to the enterprise bean class and promote them to the home and remote interfaces.
4. Add custom finders where necessary.
5. Add, define, and map any additional required CMP fields:
   a. Add any additional CMP fields to the enterprise beans.
   b. Add associations as needed.
   c. Map the CMP fields to a database schema by doing one of the following:
      - Generate a default schema and map.
      - Import a schema from an existing database table and map the CMP fields.
      - Create a schema, create a table and columns for the new schema, and map the CMP fields.
6. Set the deployment and control descriptor properties (if the default properties are not sufficient)
7. Generate the EJB deployed classes.
8. Create EJB access beans and use them to create your client application.
9. Test the enterprise beans.
   a. Create an EJB server configuration.
   b. Set the EJB server properties (if the default properties are not sufficient)
   c. Start the database servers (if testing entity beans).
   d. Create any required database tables.
   e. Start the Persistent Name Server from the WebSphere Test Environment Control Center.
   f. Start the EJB server.
   g. Run the EJB test client or run your own client to test the enterprise bean's home and remote methods.
10. Export your enterprise beans to an EJB or deployed JAR file.
11. Export your client-side code to a client JAR file.

© Copyright IBM Corp. 1998, 2000

12. Deploy the run-time JAR file for access beans and associations. (WebSphere Application Server, Advanced Edition includes this JAR file, so this step is not needed for code deployed to that platform.)

RELATED CONCEPTS

Overview of the EJB Development Environment

RELATED TASKS

Loading the required features
Adding a JDBC driver to the class path

Loading the required features

To use the EJB Development Environment, you must first load the required features.

To load the required features:

1. Start VisualAge for Java.
2. In the Workbench Projects page, select **File - Quick Start**. The Quick Start dialog appears.
3. In the left pane, select **Features**, then in the right pane, select **Add Feature** and click **OK**. The Selection Required dialog appears.
4. In the Selection Required dialog, select **IBM EJB Development Environment** and click **OK**. The IBM EJB Development Environment feature is loaded, and a number of other features are also automatically loaded that are required by the EJB Development Environment, such as the IBM WebSphere Test Environment feature.

RELATED CONCEPTS

Overview of the EJB Development Environment

RELATED TASKS

Using the EJB Development Environment: overview

Adding a JDBC driver to the class path

To run and test entity beans in VisualAge for Java, you must add a JDBC driver to the class path.

To add the DB2® JDBC driver to the class path:

1. In your file system, navigate to the following DB2 directory (where *path* is the install path of DB2):

   path\SQLLIB\java12\
2. Run the following command to change your DB2 installation to use JDBC 2.0:

   usejdbc2.bat

   (If you later find that you once again need to have your DB2 installation use JDBC1.0, you can run the usejdbc1.bat file that resides in the same location as the usejdbc2.bat file.)

3. In the Workbench Projects page, select **Window - Options**. The Options dialog appears.
4. In the tree view, select **Resources**. The Resources page appears in the dialog.
5. Beside the **Workspace class path** field, click the **Edit** button to open a class path dialog.
6. Click the **Add Jar/Zip** button and navigate to the following directory:

   drive\SQLLIB\java
7. Select the JDBC driver file db2java.zip and click **Open**.
8. In the class path dialog, click **OK**.
9. In the Options dialog, click **OK**.

RELATED CONCEPTS

Overview of the EJB Development Environment

RELATED TASKS

Using the EJB Development Environment: overview

Chapter 3. Adding and importing EJB groups and enterprise beans

Adding EJB groups

An EJB group is a logical group that allows you to organize your enterprise beans. You can perform global operations on an EJB group that will iterate on all of the enterprise beans that reside in the group. For example, if you select an EJB group to export to an EJB JAR file, all of the enterprise beans contained in the group will be exported.

There are two ways to add an EJB group to the EJB page:

- Create a new EJB group from scratch or retrieve one or more existing EJB groups from the repository.
- Create a new EJB group from an existing schema or Persistence Builder model.

This topic discusses how to create an EJB group from scratch or retrieve an EJB group from the repository. Information about creating a new EJB group from an existing schema or model is found in the topic "Creating EJB groups from schemas or models".

To add EJB groups:

1. In the Workbench Projects page, from the **Selected** menu, choose **Add - Project**. The Add Project SmartGuide appears.
2. In the **Create a new project named** field, type in the name you want to assign to the new project that will contain your enterprise beans, then click **Finish**. The new project appears in the Projects page.

   When creating projects, we suggest that you create a project to hold the enterprise beans and their related classes, and another project to hold the client application code that will access the enterprise beans. This helps to keep your code organized.
3. In the Workbench Projects page, click on the **EJB** tab. The EJB page appears.
4. From the **EJB** menu, select **Add - EJB Group**. The Add EJB Group SmartGuide appears.
5. Beside the **Project** field, click the **Browse** button and select the name of the project that you want to contain the EJB group, then click **OK**.
6. Do one of the following:

   - To add an EJB group by creating a new group, ensure that the **Create a new EJB group named** radio button is selected, and type the name you want to assign to the new group.
   - To add one or more EJB groups by retrieving existing EJB groups from the repository, do the following:

     a. Select the **Add EJB group(s) from the repository** radio button.
     b. Select an existing EJB group and an associated edition number from the **Available group names** and **Available editions** panes.
     c. Repeat the previous step for each EJB group you want to add.

© Copyright IBM Corp. 1998, 2000

7. Click **Finish**. The EJB group is added to the Enterprise Beans pane. The following EJB group icon appears next to the EJB group name:

RELATED CONCEPTS

Overview of the EJB Development Environment

RELATED TASKS

Using the EJB Development Environment: overview
Creating EJB groups from schemas or models
Adding enterprise beans
Importing enterprise beans from a JAR file

Creating EJB groups from schemas or models

To employ the bottom-up approach to mapping enterprise beans to database tables, you can create a default EJB group from a schema or from a Persistence Builder model. (Additional information about the bottom-up approach is found in the topic "Approaches to mapping enterprise beans to database tables.")

If you create a default EJB group from a schema, a mapping is also generated that maps the default EJB group to the schema.

If you create a default EJB group from a Persistence Builder model that is currently mapped, the new EJB group will also be mapped. A copy of the original map is created with the same name as that of the EJB group. This copy maps the EJB group to the original schema.

Before you create a default EJB group from a schema, check the logical names being used. VisualAge for Java uses the logical names to form names for the Java classes that are generated for the EJB group. Make sure that all characters used in these names will produce valid Java class names. In addition, the WebSphere Application Server does not support class or package names that include double-byte characters.

Before you can create a default EJB group from a model, you must load the Persistence Builder feature into the workspace. (For information about how to load the Persistence Builder feature, you can follow the instructions in the topic "Loading the required features", but ensure that you load the *Persistence Access Builder* feature rather than the EJB Development Environment feature.)

To create a default EJB group from a schema or model, do the following:

1. Import the schema or model into the EJB Development Environment.
   - To import a schema, use the Schema Browser. (Information about importing schemas is found in the topic "Creating schemas.")
   - To import a model, do the following:
     a. Add the model's storage class to the workspace.
     b. Open the Model Browser from the Workbench by selecting **Workspace > Tools > Persistence Builder Tools > Browse Models**.
     c. From the models menu of the Model Browser, select **Load Available Models**.

2. In the EJB page of the Workbench, select **EJB > Add > EJB Group from Schema or Model**. The **Create EJB Group from Schema or Model** SmartGuide appears.
3. In the **Project** field, type the name of the project that you want to contain the default EJB group. Alternatively, you can click **Browse** and select a project.
4. In the **Package** field, type the name of the package that you want to contain the default EJB group. Alternatively, you can click **Browse** and select a package. This is the package where all generated classes will be placed.
5. In the **Create a new EJB group named** field, type the name that you want to give to your new default EJB group. If you are creating a default EJB group from a schema, the name that you give to the EJB group will also be used for the EJB group map that is created to map the new EJB default group to the selected schema. (Note that you cannot specify a name for a group that is already being used by another group. You also cannot specify a name for a group that is already being used by a map, because both the group and the map would have the same name.)
6. Do *one* of the following:
   - If you are creating a default EJB group from a schema, ensure that the **Create from schema** radio button is selected. From the **Available Schemas** list, select the schema from which you want to create the default EJB group. (This list contains all of the schemas that are currently loaded in the EJB Development Environment.)
   - If you are creating a default EJB group from a Persistence Builder model, select the **Create from model** radio button. From the **Available Persistence Builder Models** list, select the model from which you want to create the default EJB group. The list of Persistence Builder models contains all of the models that are currently loaded in the Persistence Builder Model browser.
7. Click **Finish** to generate the EJB group.

   If you are creating the default EJB group from a schema, the EJB group will contain CMP entity beans for each table that exists in the selected schema. A one-to-many association will also be created for each foreign-key relationship in the schema. An EJB group map will exist in the Map Browser that maps each schema table to its corresponding CMP entity bean.

   If you are creating the default EJB group from a Persistence Builder model, the EJB group will contain a CMP entity bean for each Persistence Builder class that exists in the selected model. Each attribute from a Persistence Builder class will become a CMP field on its corresponding CMP entity.

RELATED CONCEPTS

Overview of the EJB Development Environment

RELATED TASKS

Loading the required features
Using the EJB Development Environment: overview
Approaches for mapping enterprise beans to database tables
Creating schemas

RELATED REFERENCES

Type mappings for bottom-up programming

Adding enterprise beans

Once you have added one or more EJB groups to the EJB page, you can add either session or entity enterprise beans to the EJB groups in one of the following ways:

- Creating new enterprise beans
- Retrieving existing enterprise beans from the repository
- Importing enterprise beans from EJB JAR files

This topic discusses how to add enterprise beans by creating new enterprise beans or by retrieving existing enterprise beans from the repository. Information about adding enterprise beans by importing them from a JAR file is found in the topic "Importing enterprise beans from a JAR file."

After you have added enterprise beans to an EJB group, the Enterprise Beans pane displays both the enterprise bean name and an associated icon that identifies whether the enterprise bean is a session bean, a bean-managed persistence (BMP) entity bean, or a container-managed persistence (CMP) entity bean:

- The  symbol identifies a session bean
- The  symbol identifies a BMP entity bean
- The  symbol identifies a CMP entity bean

An enterprise bean consists of a number of Java classes and interfaces, such as the enterprise bean class (which is a Java class that implements a javax.ejb.EntityBean or a javax.ejb.SessionBean class), the remote interface, the home interface, and a set of deployed classes. You need to manage all these classes as a single entity, so that they can all exist in synchronization. The tools help you keep these types synchronized. For example, if you delete a method from the enterprise bean class, the corresponding method in the remote interface will be deleted automatically.

To manage all of the Java classes and interfaces that belong to a bean as a single object, VisualAge for Java allows you to create enterprise beans as first-class objects. In the Create Enterprise Bean SmartGuide, you can define a name for the enterprise bean and then define the individual classes and interfaces that belong to the enterprise bean. You can then version the enterprise bean as a single entity. For example, when you version an enterprise bean, it will version all of the corresponding Java classes and interfaces and save their signatures into the enterprise bean object. When a specific version of the enterprise bean is loaded, the correct version of all the enterprise bean classes and remote and home interfaces will be loaded.

If you attempt to add an enterprise bean to the EJB page, but you lack sufficient authority, the enterprise bean will be added, but its association Java classes and interfaces will not be added. You should delete this partial bean and fix the authority or ownership problem, then you will be able to add the bean successfully.

Tip: To simplify the managing and versioning of your enterprise beans and to keep your enterprise bean code synchronized, it is recommended that you keep your enterprise bean source code and associated packages in the same project.

Opening the Create Enterprise Bean SmartGuide
To open the Create Enterprise Bean SmartGuide:

1. In the Enterprise Beans pane, select an EJB group to contain your enterprise beans and click mouse button 2. (Information about creating EJB groups is found in the topic "Adding EJB groups.")
2. From the pop-up menu, select **Add > Enterprise Bean**. The Create Enterprise Bean SmartGuide appears.

Using the Create Enterprise Bean SmartGuide, you can add an enterprise bean to an EJB group by either creating a new enterprise bean or by retrieving one or more existing enterprise beans from the repository. These two approaches to adding an enterprise bean to an EJB group are described in the two sections below entitled "Creating a new enterprise bean" and "Retrieving existing enterprise beans from the repository."

Creating a new enterprise bean

To add an enterprise bean to an EJB group by creating a new enterprise bean:

1. On the Create Enterprise Bean page of the SmartGuide, ensure that the **Create a new enterprise bean** radio button is selected.
2. In the **Bean name** field, type in the name that you want to assign to the enterprise bean. (The use of names with double-byte Unicode characters is not supported for enterprise beans, classes associated with enterprise beans, or packages.)
3. You can create an enterprise bean by either creating a new enterprise bean class or by using an existing enterprise bean class. (An enterprise bean class is a Java class that implements a javax.ejb.EntityBean or javax.ejb.SessionBean class.) If you want to create a new enterprise bean class, then in the **Bean type** drop-down list, select the type of enterprise bean that you want to create from the following entries:

   ```
   Session bean
   Entity bean with bean-managed persistence (BMP) fields
   Entity bean with container-managed persistence (CMP) fields
   ```

4. Do one of the following:

   - If you are creating an enterprise bean by creating a *new* enterprise bean class, then select the **Create a new bean class** radio button.
   - If you are creating an enterprise bean by using an *existing* enterprise bean class, select the **Use an existing bean class** radio button. (When you are creating an enterprise bean by using an existing enterprise bean class, you do not need to specify the type of enterprise bean that you want to create by selecting an entry from the **Bean type** drop-down list. The SmartGuide automatically determines the correct type of enterprise bean by examining the existing enterprise bean class that is specified in the **Class** field.)

     (Note that sharing a bean class or interfaces between CMP entity beans is not recommended. If, for example, you attempt to share a bean class between two CMP entity beans when defining an association, you will encounter problems. You will also encounter problems if you try to share home or remote interfaces when defining associations.)

5. The **Project** field shows the name of the current project that is associated with the selected EJB group. Do one of the following:

- If the project that appears in the **Project** field is the project that contains (or will contain) the enterprise bean class, then accept the project name without modification.
- If the project that appears in the **Project** field is *not* the project that contains (or will contain) the enterprise bean class, then in the **Project** field, type in the name of the project that you want contains (or will contain) the enterprise bean class. (If the project you specify in the **Project** field does not yet exist, it will automatically be created when the enterprise bean is created.)

  Alternatively, you can click **Browse** to select the project from a list of those projects that reside in the workspace.

6. The **Package** field shows the name of the current package that is associated with the selected EJB group. Do one of the following:

   - If the package that appears in the **Package** field is the package that contains (or will contain) the enterprise bean class, then accept the package name without modification.
   - If the package that appears in the **Package** field is *not* the package that contains (or will contain) the enterprise bean class, then in the **Package** field, type in the name of the package that contains (or will contain) the enterprise bean class. (If the package you specify in the **Package** field does not yet exist, it will automatically be created when the enterprise bean is created.)

     Alternatively, you can click **Browse** to select the package from a list of those packages that reside in the specified project.

     (The package holds implementation types for the enterprise bean, such as the remote and home interfaces, enterprise bean class and key classes, and stubs and ties. However, the bean class, remote interface, home interface, and other elements may reside in different packages if you specify fully qualified names for each of them. If the packages specified in the fully-qualified names do not exist in the workspace, they will be created automatically. In summary, the package specified in the **Package** field is only used if a fully-qualified name is not used for the bean class, remote interface, home interface, etc.)

7. Do one of the following:

   - If you are creating an enterprise bean by creating a *new* enterprise bean class, in the **Class** field, either accept the default value or type in a new name for the class. (It is recommended that you accept the default value so that all of the implementation types of the enterprise bean are named properly.)
   - If you are creating an enterprise bean by using an *existing* enterprise bean class, in the **Class** field, either type in the name of the enterprise bean class or click the **Browse** button and select the enterprise bean class from the list of classes contained in the specified package.

   Notes: (1) If the name specified in the **Class** field is a Java simple name, the SmartGuide assumes that the class exists (or will exist) in the package specified in the **Package** field. (2) If an existing package is specified in the **Package** field, clicking **Browse** beside the **Class** field shows the enterprise

bean classes in the existing package. (3) If the **Package** field is empty or contains a package name that does not exist, clicking **Browse** beside the **Class** field shows all the enterprise bean classes in the workspace. Once you select the enterprise bean class you want, the **Project** and **Package** fields are updated for the enterprise bean class you selected.

8. If you are creating an enterprise bean by creating a new enterprise bean class, beside the **Superclass** field, click **Browse** and select the existing class that you want to use as a superclass for your new enterprise bean class.

   Note that for the superclass, you must specify an existing Java class. This existing Java class does not necessarily need to be an enterprise bean class. However, if you specify an existing enterprise bean class as the superclass, the **Bean type** field is updated to be the bean type of the superclass. You will not be able to change this. Therefore, if you intend to extend an existing enterprise bean class, you only need to specify a name for your new enterprise bean in the **Bean name** field and then specify the superclass in the **Superclass** field. The **Bean type** field will be filled in automatically with the correct bean type.
9. Do one of the following:

   - If you want to accept the default names that will be assigned to the home and remote interfaces and key class, and you do not want to add, change, or remove any CMP fields, superinterfaces, or import statements, click **Finish** to immediately generate the enterprise bean. The new enterprise bean appears in the Enterprise Beans pane under the selected EJB group. The classes and interfaces associated with the new enterprise bean appear in the Types pane.
   - If you want to change the default names that will be assigned to the home or remote interfaces or key class, or if you want to add, change, or remove any CMP fields, superinterfaces, or import statements, click **Next** to go to the next page of the SmartGuide. The Define Bean Class Attributes and Interfaces page appears.
10. If you want to change the default name of the home interface, remote interface, or key class (entity beans only), then in the **Home interface**, **Remote interface**, or **Key class** field, type in a new name. Alternatively, you can click the **Browse** button beside any of these fields to select the name of an existing home interface, remote interface, or key class.
11. If you are creating a CMP entity bean and you intend to add additional finder methods other than the findByPrimaryKey() method, ensure that the **Create finder helper interface to support finder methods** check box is selected. You should only deselect the check box if you do not plan to add any extra finder methods. (Note that the finder helper interface is useful only if you have extra finder methods.)
12. If you are creating a CMP entity bean and you want to add, edit, or remove a CMP field, do one of the following:

    - If you want to add a new CMP field, beside the **Add CMP fields to the bean** list box, click **Add**. The Create CMP Field SmartGuide opens. (For more information about using the Create CMP Field SmartGuide, see the topic “Adding and editing CMP fields.”)
    - If you want to edit a CMP field, select the CMP field in the **Add CMP fields to the bean** list box, then click **Edit**. The Edit CMP Field SmartGuide opens. (For more information about using the Edit CMP Field SmartGuide, see the topic “Adding and editing CMP fields.”)

- If you want to remove a CMP field, select the CMP field in the **Add CMP fields to the bean** list box, then click **Remove**.

13. If you want to add an interface for the remote interface to extend, or if you want to remove an interface that is extended by the remote interface, do one of the following:

    - If you want to add an interface for the remote interface to extend, then beside the **Which interfaces should the remote interface extend?** lsit box, click **Add** and select the interface that you want to extend.
    - If you want to remove an interface that the remote interface extends, then in the **Which interfaces should the remote interface extend?** list box, select the interface that you want to remove and click **Remove**.

14. If you want to add or remove an import statement for the enterprise bean class, do one of the following:

    - If you want to add a class or interface to an import statement, then beside the **Add import statements to the bean class** list box, click **Add Type** and select the class or interface that you want to add.
    - If you want to add a package to an import statement, then beside the **Add import statements to the bean class** list box, click **Add Package** and select the package that you want to add.
    - If you want to remove a class, interface, or package from the import statement, then in the **Add import statements to the bean class** list box, select the class, interface, or package that you want to remove and click **Remove**.

15. Click **Finish**. The new enterprise bean appears in the Enterprise Beans pane under the selected EJB group. The classes and interfaces associated with the new enterprise bean appear in the Types pane.

Retrieving existing enterprise beans from the repository
To add one or more enterprise beans to an EJB group by retrieving existing enterprise beans from the repository:

1. On the Create Enterprise Bean page of the SmartGuide, select the **Add enterprise beans from the repository** radio button.
2. Select an existing enterprise bean and an associated edition number from the **Available enterprise beans** and **Available editions** panes.
3. Repeat the previous step for each enterprise bean that you want to add.
4. Click **Finish**. The enterprise beans are loaded from the repository and appear in the Enterprise Beans pane under the selected EJB group. The classes and interfaces associated with the enterprise beans appear in the Types pane.

Note that if you are working in a team development environment and you want to load an enterprise bean from the repository, you need to be a developer of the EJB group and package to which the enterprise bean classes and interfaces belong.

RELATED CONCEPTS

Overview of the EJB Development Environment
Editions and versioning
Key classes and key fields

RELATED TASKS

Adding enterprise beans with inheritance

Using the Create Enterprise Bean with Inheritance SmartGuide, you can create a new child enterprise bean that inherits from an existing parent enterprise bean. If you create a new CMP entity bean, support is provided for single-table and root/leaf table mapping and the appropriate persister/finder code is generated to extract the appropriate instances as determined by the mapping.

Note: Support for inheritance has been included in VisualAge for Java as a technical preview. It is meant for deployment only with the WebSphere Application Server, Advanced Edition. Also, due to a limitation of the EJB specification, an enterprise bean (or other remote object) cannot inherit from two interfaces that have methods with the same name, even if the methods each have different signatures. If you encounter this problem, simply rename your methods so that each method has a unique name.

To create an enterprise bean with inheritance:

1. In the Enterprise Beans pane, select the parent enterprise bean that you want your new child enterprise bean to inherit from.
2. From the **EJB** menu, select **Add** > **Enterprise Bean with Inheritance**. The Create Enterprise Bean with Inheritance SmartGuide appears.
3. In the **Bean name** field, type the name that you want to assign to the new enterprise bean. (As you type in a name for the bean, default names that are based on the bean name appear in the **Bean class**, **Remote interface**, and **Home interface** fields.)
4. In the **Inherit from** drop-down list box, ensure that you see the name of the existing parent enterprise bean that you want your new enterprise bean to inherit from. (If you want to specify a different parent enterprise bean, you can select one from the drop-down list.)
5. In the **Package** field, ensure that you see the correct package name.

   If you want, you can type in a different package name or click the **Browse** button to select the package that will contain the enterprise bean class, then click **OK**. If you enter a package that does not exist, a new package is created.

   (Essentially, the package holds implementation types for the enterprise bean, such as the remote and home interfaces, enterprise bean and key classes, and stubs and ties. However, the bean class, remote interface, home interface, and other elements may reside in different packages if you specify fully qualified names for each of them. If the packages specified in the fully qualified names do not exist in the workspace, they will be created automatically. In summary, the package specified in the **Package** field is only used if a fully qualified name is not used for the bean class, remote interface, home interface, etc.)
6. In the **Bean class** field, either accept the default name for the bean class or type in a bean class name of your own choosing.

The bean class of the new child enterprise bean will extend the bean class of the parent enterprise bean. No parent bean class methods will be defined in the child bean class.

7. In the **Remote interface** field, either accept the default name for the remote interface or type in a remote interface name of your own choosing.

   The remote interface of the new child enterprise bean will extend the remote interface of the parent enterprise bean, but it will not initially contain any methods.
8. In the **Home interface** field, either accept the default name for the home interface or type in a home interface name of your own choosing.

   Home interface classes cannot inherit from other home interface classes. When an inheritance relationship is defined between enterprise beans, all home methods that are not part of the association are copied to the child bean. The child interface will still only extend javax.ejb.EJBHome.
9. If you are creating a CMP entity bean and you intend to add additional finder methods other than findByPrimaryKey(), ensure that the **Create finder helper interface to support finder methods** check box is selected. You should only clear the check box if you do not plan to add any extra finder methods. (Note that the finder helper interface is useful only if you have extra finder methods.)
10. If you are creating a CMP entity bean and you want to add, edit, or remove a CMP field, do one of the following:

    - If you want to add a new CMP field, beside the **Add CMP fields to the bean** list box, click **Add**. The Create CMP Field SmartGuide opens. (For more information about using the Create CMP Field SmartGuide, see the topic "Adding and editing CMP fields.")
    - If you want to edit a CMP field, select the CMP field in the **Add CMP fields to the bean** list box, then click **Edit**. The Edit CMP Field SmartGuide opens. (For more information about using the Edit CMP Field SmartGuide, see the topic "Adding and editing CMP fields.")
    - If you want to remove a CMP field, select the CMP field in the **Add CMP fields to the bean** list box, then click **Remove**.
11. If you want to add an import statement to the new bean class, do one of the following:

    - If you want to add an import statement to the new bean class and import a class or interface, beside the **Add import statements to the bean class** list box, click **Add Type** and select the class or interface that you want to import.
    - If you want to add an import statement to the new bean class and import a package, beside the **Add import statements to the bean class** list box, click **Add Package** and select the package that you want to import.

    If you later want to remove an import statement from the new bean class, in the **Add import statements to the bean class** list box, select the import statement that you want to remove and then click **Remove**.
12. Click **Finish** to generate the enterprise bean.

Note that if you are adding an enterprise bean and it inherits from a CMP entity bean, a validation error will exist in the table maps if both of the following conditions exist for the CMP entity bean:

- It did not previously have any inherited beans.
- It is already mapped with a "no inheritance" table map (which could have been created using the top-down approach).

You will either need to delete the table map that has an error and then regenerate the maps, or you can rework the table map manually to eliminate the errors (editing by hand is required if you want a root/leaf mapping). After the inheritance mapping (single table or root/leaf) is established, subsequent top-down map additions will have reasonable results. Information about generating maps is found in the topic "Creating EJB group maps".

RELATED CONCEPTS

Overview of the EJB Development Environment

RELATED TASKS

Using the EJB Development Environment: overview
Adding EJB groups
Adding enterprise beans
Adding and editing CMP fields
Creating EJB group maps

Adding enterprise beans for the Enterprise Access Builder

In the EJB Development Environment, you can create specialized session and BMP entity enterprise beans that are intended for use with the Enterprise Access Builder (EAB) component. To help you edit or create session beans for the EAB component, an editor and a Create Session Bean SmartGuide is provided. You must have the EAB component installed and loaded in the VisualAge for Java workspace before you can work with this particular SmartGuide.

Information about editing or creating EAB session beans is found in the online documentation for the EAB component.

RELATED TASKS

Adding EJB groups
Adding enterprise beans

Importing enterprise beans from a JAR file

Once you have added one or more EJB groups to the EJB page, you can add enterprise beans to the EJB groups by:

- Importing enterprise beans from EJB JAR files
- Creating new enterprise beans
- Retrieving existing enterprise beans from the repository

This topic discusses how to add enterprise beans by importing enterprise beans from a JAR file. Information about adding enterprise beans by creating new enterprise beans or by retrieving existing enterprise beans from the repository is found in the topic "Adding enterprise beans." Information about importing enterprise beans into a repository is found in the topic "Moving or copying enterprise beans between repositories."

Notes

- If you want to export and later import enterprise beans that exist in an inheritance or association relationship, you must export the project containing the beans into a repository (.dat) file to preserve the information that defines the inheritance or association. When you later import from the .dat file, the inheritance and association information is preserved. Inheritance and association information is not preserved if you import the enterprise beans in an EJB JAR file.
- If an enterprise bean you plan to import already exists in the EJB page, it will not overwrite the existing enterprise bean.
- If an access bean with one or more copy helpers exists for an enterprise bean that is to be exported to an EJB JAR file, then the class com.ibm.ivj.ejb.runtime.CopyHelper is also exported to the JAR file. If you later delete the enterprise bean from the workspace and then import it again in an EJB JAR file, you will be prompted about whether you want to create an open edition for com.ibm.ivj.ejb.runtime.CopyHelper. You should specify that you do *not* want to create an open edition.
- If you import a .dat file but you can't find the enterprise beans it contains, it may be because you did not version the entire project containing your beans before you exported it, or you didn't load the entire project when you imported the .dat file. If you cannot see expected EJB groups in the EJB page after loading the entire project, it is probably because the corresponding reserved package is not included in the loaded version of the project. You should always add EJB groups in the EJB page, rather than adding the reserved package in the Projects page. If, however, you have already added the reserved package in the Projects page, you will not be able to add the EJB group in the EJB page because the group will not show in the "Add EJB group(s) from repository" list box. The workaround is to go to the Projects page and delete the reserved package. Then the groups will show in the "Add EJB group(s) from repository" list box.

To import enterprise beans from a JAR file:

1. In the Enterprise Beans pane of the EJB page, select the EJB group that you want to contain the imported enterprise beans.
2. From the pop-up menu, select **Import Enterprise Beans**. The Import from an EJB JAR File SmartGuide opens.
3. Beside the **Filename** field, type the name of the JAR file that you want to import.

   Alternatively, click the **Browse** button to locate the JAR file. The SmartGuide marks for importation all of the types and beans found in the JAR file.
4. Select the file types that you want to import. (By default, all the files for any selected file type are imported. Click the **Details** button next to a file type to see a list of the files that will be imported and to specify individual files to import.)
5. Specify additional options as appropriate using the **Options** check boxes.
6. Click **Finish**.

RELATED CONCEPTS

Overview of the EJB Development Environment

RELATED TASKS

Using the EJB Development Environment: overview
Adding enterprise beans
Moving or copying enterprise beans between repositories

Chapter 4. Adding home and remote methods

Customizing ejbCreate and ejbPostCreate methods

When you use the Create Enterprise Bean SmartGuide to create an enterprise bean, templates are automatically created for the ejbCreate and ejbPostCreate methods. You may, however, need to fill in the method bodies or add new ejbCreate and ejbPostCreate methods. You may also need to initialize all of the CMP fields in the ejbCreate method.

If you want to create a new ejbCreate or ejbPostCreate method, you can choose an existing ejbCreate or ejbPostCreate method and override the method signature. A new method will be created and your existing method will not be lost.

Writing ejbCreate methods with required single-valued roles

In an ejbCreate method for an enterprise bean with a required association role, it is essential for the foreign-key fields to be set before the ejbCreate method exits. Otherwise, database exceptions can result as NULLs and be written to nonNULLable columns. For example, if Employee has the required role *department*, you can write an ejbCreate method like this:

```
public void ejbCreate(String argEmpId, Department argDepartment)
                      throws javax.ejb.CreateException,
                             java.rmi.RemoteException {
   _initLinks();

// All CMP fields should be initialized here.
   empId = argEmpId;
   setDepartment(argDepartment);
}
```

You could also pass in a DepartmentKey instance as the parameter, but this is discouraged because:

- The key might not be valid if it does not come directly from a known entity.
- This practice does not enable you to take advantage of the association maintenance capability of the association runtime within the current transaction (it would be handled by the association maintenance runtime in subsequent transactions).

In order to take advantage of the association maintenance runtime in the current transaction, you must also write an ejbPostCreate method that looks like this:

```
public void ejbPostCreate(String argEmpId, Department argDepartment)
                          throws java.rmi.RemoteException {
   setDepartment(argDepartment);
}
```

In addition, you will need to add the create method to the home interface. You can do this by selecting the ejbCreate method on the **Members** pane of the EJB page in the IDE and selecting **Add To > EJB Home Interface** from the pop-up menu.

This Employee-Department example was a 1:M association with Employee as the 1 end and holding the foreign key. For 1:1 associations, either end can hold the foreign key. For the creation of the foreign-key holder, this same creation pattern will work. For the end that does not hold the foreign key, the same pattern will work, but only the set call in the post-create method is actually needed.

© Copyright IBM Corp. 1998, 2000

RELATED CONCEPTS

Overview of the EJB Development Environment

RELATED TASKS

Using the EJB Development Environment: overview
Adding enterprise beans
Implementing custom finders in home interfaces for CMP entity beans
Adding remote methods

Adding home methods

When you create an enterprise bean in the EJB Development Environment, a set of required methods for the bean is automatically created. If you select the enterprise bean in the Enterprise Beans pane, the Java methods of the bean implementation class are displayed in the Members pane. You can modify these required methods. In addition, you can add two other types of methods to the home interface: ejbCreate...( ) and ejbFind...( ).

Modifying methods or adding methods in the EJB Development Environment is similar to modifying or adding methods elsewhere in VisualAge for Java.

Once you modify the methods or add a create or find method, you can promote the methods to the home interface from the bean class without editing the home interface directly. Note, however, that for CMP enterprise beans, you must define all Find methods directly in the home interface and not promote them from the enterprise bean class.

Any methods that correspond to methods declared in the home interface are identified with the following symbol:

Note that if you add a home method after generating the deployed code, error symbols will appear beside the enterprise bean and old generated code. You must regenerate the deployed code.

For the most part, you can create all of your home methods in exactly the same way. However, creating Find methods requires some additional steps.

To add a new home method:

1. In the Types pane of the EJB page, select the enterprise bean class to which you want to add a method.
2. Click mouse button 2, then select **Add > Method** from the pop-up menu. The Create Method SmartGuide appears.
3. Follow the SmartGuide instructions on creating a new method.
4. Click **Finish** to create and compile the new method.
5. If you want to promote a home method to the home interface, do the following:

    a. In the **Types** pane, select the enterprise bean class for which you want to promote a home method.
    b. In the **Members** pane, select the home method that you want to promote, such as ejbCreate or ejbFind, then click mouse button 2.

c. Select **Add To** > **EJB Home Interface**.

Note that if you add a home method and promote it to the home interface after generating the deployed code, you must regenerate the deployed code.

6. If you want to remove a home method from the home interface, do the following:

   a. In the **Types** pane, select the enterprise bean class for which you want to remove a home method.
   b. In the **Members** pane, select the home method you want to remove, such as ejbCreate or ejbFind, then click mouse button 2.
   c. Select **Remove From** > **EJB Home Interface**.

   Note that if you delete a home method in the enterprise bean class, the corresponding method in the home and remote interface is deleted automatically. Also, if you remove a method from the home interface, you should regenerate the deployed code.

7. If the new method you have created is a Finder method for a CMP bean, define a Finder method in the Home interface. Information about implementing custom finders is found in the topic "Implementing custom finders in home interfaces for CMP entity beans".

RELATED CONCEPTS

Overview of the EJB Development Environment

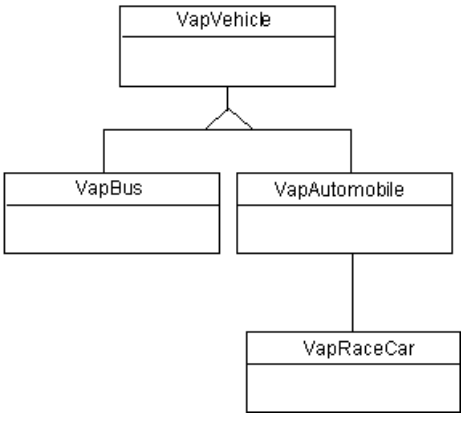
RELATED TASKS

Using the EJB Development Environment: overview
Adding enterprise beans
Implementing custom finders in home interfaces for CMP entity beans
Adding remote methods

Implementing custom finders in home interfaces for CMP entity beans

The custom finder support discussed in this section assumes that WebSphere Application Server, Advanced Edition is the deployment platform.

Consider the following hierarchy of CMP entity beans:



There is also another CMP entity bean called VapGarage. An association between VapVehicle and VapGarage exists in which a garage can hold many vehicles.

Finally, there is a CMP entity bean called VapMotorVehiclePart. An association between VapVehicle and VapMotorVehiclePart exists in which a vehicle can contain many parts.

History

Custom finder support in previous versions of the VisualAge for Java EJB tools was limited. For example, if VapGarageHome has the following finder method:

```
java.util.Enumeration
     findCapacityGreaterThan(int capacity)
        throws java.rmi.RemoteException,
               javax.ejb.FinderException;
```

the finder helper interface for VapGarage might look like this:

```
.
.
.
**/
public interface VapGarageBeanFinderHelper {
   final static String
      findCapacityGreaterThanQueryString=
      "SELECT T1.ID, T1.CAPACITY FROM SAMPLE.GARAGE T1
       WHERE T1.CAPACITY > ?";
}
```

When deployed code was generated, these two items would be matched up and a method would be created on a finder class that injected the single parameter into the SQL query. The finder would then use the generated persister to run and interpret the results of the query.

This approach had several problems:

- The queries had to be simple with straightforward parameter injection.
- If the map and schema browsers were used to map the bean, the query shape sometimes had to be modified to match the result-set shape of the generated persister. The order and number of columns would change as the bean and mapping evolved.
- Now that mapping of inheritance is supported, the queries can become very complex with multiple tables and unions of multiple subselects. This greater complexity makes errors in the query string more likely.

The following sections describe the currently supported types of custom finders, which you can use in combination.

Note that the return type java.util.Enumeration on the finder in the home interface indicates that this finder may return more than one bean. Using the remote interface and the return type indicates that a single bean is returned. This is true for all the types of custom finders discussed in this section. The code generated into the persister handles this distinction.

SELECT custom finders

A custom finder created before VisualAge for Java, Version 3.0, is called a SELECT custom finder. This is because you enter an entire select statement in the finder helper interface to define the SQL query.

The use of SELECT custom finders is supported for compatibility with earlier releases. Using SELECT custom finders is discouraged in this and future releases of VisualAge for Java. However, some changes have been made to help mitigate some of the problems previously discussed. Because the order of columns in the generated queries is now more stable, the query string should require fewer modifications.

WHERE custom finders

A custom finder in which you enter only the filtering WHERE clause into the finder helper interface is called a WHERE custom finder. Using the same garage example as before, the finder helper interface would look like this:

```
public interface VapGarageBeanFinderHelper {
   public final static String
      findCapacityGreaterThanWhereClause =
         "T1.CAPACITY > ?";
}
```

Notice that any dependency on the shape of the results is removed from the string. Two dependencies still exist:

- The name of the column (CAPACITY)
- The alias for the table (T1)

The name of the column would change only if you took action to change it.

The alias for the table will be the same from one generation to the next unless tables are added to or removed from the mapping. In single-table cases, this may not seem significant (the alias is always T1). When multiple tables are used, this is very important.

For example, consider the VapVehicle hierarchy. When it is mapped using root/leaf inheritance mapping, there is at least one table for each class in the hierarchy, and the query into which the WHERE clause would be inserted would have multiple subselects. The WHERE clause would be inserted into each subselect, and it would be valid SQL syntax for each subselect to use a different alias for the same table. However, our query generation ensures that the same table has the same alias across the entire query. If it did not, this WHERE clause substitution technique could not work. In these cases, the code generated into the persister knows to inject any finder parameters into the query multiple times.

There is one very important restriction when using this form of custom finder in a mapped hierarchy: The WHERE clause can only reference tables that map the bean in which the finder is defined or tables that map one of bean's parent beans.

For example, a WHERE clause in the VapVehicleBeanFinderHelper can only reference columns in the table used to map the VapVehicle bean. However, a WHERE in the VapAutomobileBeanFinderHelper can reference columns from either the table that maps VapAutomobile or VapVehicle (its superclass).

Any table references in your handwritten SQL code must match the table aliases set up in the genericFindSqlString field. This is declared in the enterprise bean's generated persister.

As in the SELECT custom form, the number of finder parameters must match the number of injection points (the **?** characters) in the WHERE clause. Also, as in the SELECT form, the type of the parameter will be used to determine which java.sql.PreparedStatement set call will be used to inject each parameter. The

parameter types must be compatible with the column types. If the type is an object type and the parameter is null, a setNull call will be used.

For each finder method that is defined in the home interface (other than findByPrimaryKey and those finder methods generated to support associations), an SQL query string or a method declaration must be defined in the interface. For example, the home interface may contain the following method:

```
public java.util.Enumeration findGreaterThan (int threshold) throws
   java.rmi.RemoteException, javax.ejb.FinderException;
```

In this interface, the WHERE custom finder is one of the 3 forms that you can provide. For example (line broken for publication):

```
public static final String findGreaterThanWhereClause =
    "T1.VALUE > ?";
```

Note, however, that if you have an SQL statement that contains no WHERE clause, such as `SELECT * FROM MYTABLE`, you should use a query string that always evaluates to *true*. For example:

```
public static final String findALLWhereClause = "1 = 1";
```

The WHERE custom finder form solves many of the problems of the SELECT custom finder form, except the parameter to query mapping is still limited to simple and straight forward substitutions. The method custom finder form solves that problem when maximum flexibility is needed.

Method custom finders

A custom finder in which you enter a method signature into the finder helper interface is called a method custom finder. It is the most flexible type of custom finder, but it requires more work on your part. Using the same garage example as before, the finder helper interface would look like this:

```
public interface VapGarageBeanFinderHelper {
  public java.sql.PreparedStatement findCapacityGreaterThan(int threshold)
       throws Exception;
}
```

Of course, as in the SELECT and WHERE forms, this is not enough. An implementation of this method is needed. Also, the implementation is going need help from the persister to make sure the result set for the query has the correct shape. You provide your implementation of the method in a class that follows these rules:

- The name of the class is either *beanName*BeanFinderObject (VapGarageBeanFinderObject, in this example) and is in the same package as the bean class, or the name of the class is specified in the environment properties of the enterprise bean. The name of the property must be **CustomFinderClassName**. The value of the property must be the fully qualified class name.
- The class must extend com.ibm.vap.finders.VapEJSJDBCFinderObject and must implement the bean's finder helper interface.
- Any table references in your handwritten SQL code must match the table aliases set up in the genericFindSqlString field. This is declared in the enterprise bean's generated persister.

The com.ibm.vap.finders.VapEJSJDBCFinderObject base class provides several important helper methods as described in the following table:

Method	Description
getMergedPreparedStatement	Takes a WHERE clause and returns a PreparedStatement with the WHERE clause merged into the appropriate places. The PreparedStatement will have the correct result set shape.
getMergedWhereCount	Returns the number of times the WHERE clause is merged into the PreparedStatement. This is needed to know how many times to inject your query parameters into the PreparedStatement.
getPreparedStatement	Takes a complete query string and returns a PreparedStatement. This can be used if for some reason you need to do your own WHERE clause merging. This should be a very rare case. The next two functions are provided to help in these extreme cases.
getGenericFindSqlString	Returns the query string into which WHERE clauses are merged.
getGenericFindInsertPoints	Returns an array of integers that defines the point or points in the getGenericFindSqlString returned query string at which the WHERE clause is merged. The first point in the array is the last point in the string. It is usually best to merge from the end of the query string since a merge at the end will not change the location of merges earlier in the string. The size of the array is the same as the value returned from getMergedWhereCount.

To finish off our example, the finder object would look like this.

```
/**
* Implementation class for methods in
* VapGarageBeanFinderHelper.
*/
public class VapGarageBeanFinderObject extends
   com.ibm.vap.finders.VapEJSJDBCFinderObject implements
   VapGarageBeanFinderHelper {

   public java.sql.PreparedStatement
   findCapacityGreaterThan(int threshold)
   throws Exception {

      PreparedStatement ps = null;
      int mergeCount = getMergedWhereCount();
      int columnCount = 1;
      ps = getMergedPreparedStatement("T1.CAPACITY > ?");
      for (int i=0; i<(columnCount*mergeCount); i=i+columnCount) {
          ps.setInt(i+1, threshold);
      }
      return ps;
   }
}
```

This is a rather simple case that can better be handled by a WHERE custom finder. More complex examples are possible that a WHERE custom finder simply could not handle. For example, suppose you wanted a finder that took a more complex object and injected it into multiple columns in a WHERE clause. You could end up with a finder method that looked like this:

```
public java.sql.PreparedStatement
   findWithComplexObject(BigObject big) throws Exception {
```

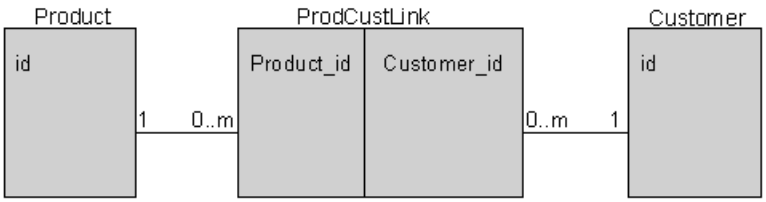
```
   PreparedStatement ps = null;
   int mergeCount = getMergedWhereCount();
   int columnCount = 3;
   int anInt = big.getAnInt();
   String aString = big.getAString();
   String aLongAsString =
      com.ibm.vap.converters.VapStringToLongConverter.
      singleton().dataFrom(big.getLongObject());

   ps = getMergedPreparedStatement("(T1.ANINT > ?) AND
        (T1.ASTRING = ?) AND (T2.ALONGSTR <  ?)");
   for (int i=0; i<(columnCount*mergeCount); i=i+columnCount) {
       ps.setInt(i+1, anInt);
       if (aString == null)
          ps.setNull(1, java.sql.Types.VARCHAR);
       else
          ps.setString(i+2, aString);
       if (aLongAsString == null)
          ps.setNull(1, java.sql.Types.VARCHAR);
       else
          ps.setString(i+3, aLongAsString);
   }
   return ps;
}
```

Even more complex examples are possible. For example, an object could be passed that contains the WHERE clause (or instructions on how to create it) in addition to the data. Or, there could be multiple parameters, each representing different conditions in the WHERE clause.

In the case of any method custom finder, the generated persister uses your implementation to create the PreparedStatement to be executed. The persister will execute the PreparedStatement and handle the results.

Example: Complex method custom finder

The following example involves a many-to-many association between Product and Customer beans, using a intermediary bean (ProdCustLink) and two 1:m associations:



You can write method custom finders to span the relationship in either direction with just one method call. For this example, consider one direction only: a finder in Customer that retrieves all Customer instances that are associated with a given product key.

Customer's home interface contains the appropriate method signature, as follows:

```
java.util.Enumeration
   findCustomersByProduct(prod.cust.code.ProductKey inKey)
   throws java.rmi.RemoteException, javax.ejb.FinderException;
```

Customer's finder helper interface contains the signature for the corresponding finder method:

```
public java.sql.PreparedStatement
   findCustomersByProduct(prod.cust.code.ProductKey inKey)
   throws Exception;
```

The finder object (CustomerBeanFinderObject) builds and caches the query string for the finder as well as implements the finder method.

```
public class CustomerBeanFinderObject
   extends com.ibm.vap.finders.VapEJSJDBCFinderObject
   implements CustomerBeanFinderHelper {

   private String cachedFindCustomersByProductQueryString = null;
   .
   .
   .
}
```

Through lazy initialization in the finder object, the accessor method for the query-string field builds up the query string by first merging the WHERE condition into the query template and then adding a reference to the intermediate table into the FROM clause.

The first half of the accessor method uses a genericFindInsertPoints array to locate and update each WHERE clause. Then, the second half of the method counts forward from the beginning of each FROM clause, inserts the reference to the intermediate table into the query string as needed, and updates the query-string field.

```
protected String getFindCustomersByProductQueryString() {
   if (cachedFindCustomersByProductQueryString == null) {

   // Do the WHERE first
   // so that the genericFindInsertPoints are correct.
      int i;
      int[] genericFindInsertPoints = getGenericFindInsertPoints();
      StringBuffer sb = new StringBuffer(getGenericFindSqlString());
      for (i = 0; i < genericFindInsertPoints.length; i++) {
         sb.insert(genericFindInsertPoints[i],
            "(T1.id = T2.Customer_id) AND (T2.Product_id = ?)");
      }

   // Make sure to update every FROM clause.
      String soFar = sb.toString();
      int fromOffset = soFar.indexOf(" FROM ");
      while (fromOffset != -1) {
         sb.insert((fromOffset+5)," ProdCustLink T2, ");
         soFar = sb.toString();
         fromOffset = soFar.indexOf(" FROM ", (fromOffset+5));
      }
      cachedFindCustomersByProductQueryString = sb.toString();
   }
   return cachedFindCustomersByProductQueryString;
}
```

After this method call, the query string looks something like the following:

```
 SELECT <columns> FROM ProdCustLink T2, CUSTOMER T1
    WHERE((T1.id = T2.Customer_id) AND (T2.Product_id = ?))
```

Also in the finder object, the implemented finder uses the query string to create a PreparedStatement. Last but not least, the product ID value is added into each WHERE clause by using the superclass method getMergedWhereCount() in the iteration loop.

```
public java.sql.PreparedStatement
   findCustomersByProduct(ProductKey inKey)
   throws java.lang.Exception {

   // Get the full query string and make a PreparedStatement.
   java.sql.PreparedStatement ps =
      getPreparedStatement(getFindCustomersByProductQueryString());

   // Inject the product id parameter into each merged WHERE clause.
   for (int i = 0; i < getMergedWhereCount(); i++) {
      if (inKey != null)
         ps.setInt(i+1, inKey.id);
      else
         ps.setNull(i+1, 4);
   }

   return ps;
}
```

Maintaining SQL compatibility across different databases

In the case where finder methods access different databases, there are situations where the SQL syntax used by each database is different. In these situations, use the SQL extensions defined by JDBC to resolve the database differences.

For example, assume that you are developing a CMP entity bean that requires a finder method that involves a timestamp/date field. Also assume that this bean will be deployed to DB2 and Oracle databases. The problem is that the format of the timestamp/date fields in DB2 and Oracle are different, which causes difficulties in defining one WHERE clause for use with both DB2 and Oracle. The solution to this particular problem is to use the SQL Escape sequence.

RELATED CONCEPTS

Overview of the EJB Development Environment

RELATED TASKS

Using the EJB Development Environment: overview
Adding enterprise beans
Adding home methods

RELATED REFERENCES

Data-type converters supported for persistence
Finder helpers generated for deployment to Component Broker

Adding remote methods

When you create an enterprise bean in the EJB Development Environment, a set of required methods for the bean is automatically created. If you select the enterprise bean in the Enterprise Beans pane, the Java methods of the bean implementation class are displayed in the Members pane. You can modify these required methods and add new remote (business) methods of your own.

Modifying methods or adding methods in the EJB Development Environment is similar to modifying or adding methods elsewhere in VisualAge for Java.

Once you have created remote (business) methods, you can promote the methods to the remote interface from the enterprise bean class without editing the remote interface directly.

Any methods that correspond to methods declared in the remote interface are identified with the following symbol:

Note that if you add a remote method after generating the deployed code, error symbols will appear beside the enterprise bean and old generated code. You must regenerate the deployed code.

To add a new method:

1. In the Types pane of the EJB page, select the enterprise bean class to which you want to add a method.
2. Click mouse button 2, then select **Add > Method** from the pop-up menu. The Create Method SmartGuide appears.
3. Follow the SmartGuide instructions on creating a new method.
4. Click **Finish** to create and compile the new method.
5. If you want to promote a method to the remote interface, select the method and click mouse button 2, then select **Add To> EJB Remote Interface**.

   Note that if you add a remote method and promote it to the remote interface after generating the deployed code, you must regenerate the deployed code. Also, if you remove a method from the remote interface, you should regenerate the deployed code.
6. If you want to remove a method from the remote interface, select the method and click mouse button 2, then select **Remove From > EJB Remote Interface**. Note that if you delete a remote method in the enterprise bean class, the corresponding method in the home and remote interface is deleted automatically.

RELATED CONCEPTS

Overview of the EJB Development Environment

RELATED TASKS

Using the EJB Development Environment: overview
Adding enterprise beans
Adding home methods

Chapter 5. Managing CMP fields and associations

Adding or editing CMP fields

When you initially create a CMP entity bean, some fields are automatically generated in the bean class. However, if you later decide that you want to add a CMP field to the entity bean and perhaps define it as a key field, you can use the Create CMP Field SmartGuide. For example, if you created a CMP entity bean named *Customer*, you might later want to add a CMP field to the entity bean that is named *Address*.

You can also change the properties for an existing CMP field by using the Edit CMP Field SmartGuide.

To display the existing CMP fields for a CMP entity bean:

1. Select the CMP bean in the Enterprise Beans pane.
2. Toggle to the Properties pane by clicking the **Fields** icon  in the upper right corner of the Types pane. To toggle back to the Types pane, click the **Types**  icon.

To add or edit a CMP field:

1. Do one of the following:
   - If you want to create a new CMP field for an existing CMP entity bean, select the CMP bean in the Enterprise Beans pane, then right click and select **Add > CMP Field**. The Create CMP Field SmartGuide opens.
   - If you want to edit an existing CMP field for a CMP entity bean, select the CMP bean in the Enterprise Beans pane and toggle to the Properties pane as described above, then select the CMP field and choose **Edit** from the pop-up menu.
2. In the **Field Name** field, specify a name for the CMP field that is unique for the entity bean class.
3. In the **Field Type** field, specify a type for the CMP field either by typing in the name of the type or by selecting a type from the drop-down list. (The drop-down list contains a list of primitive types and types that reside within the associated package.) You can also click **Browse** to select types that reside outside the associated package but within the VisualAge for Java workspace.)
4. If you want the CMP field to be an array, select the **Array** checkbox, then in the **Dimensions** spin box, specify the number of dimensions for the array.
5. To specify an initial value for the CMP field, type the value in the **Initial Value** field. For simple string types, the value must be enclosed in double quotation marks; for example: `"string-type"`.

   For an array, the value must correctly use braces; for example:

   - `{1,2,3}` for a 1-dimension array
   - `{{1,2,3}, {4,5}}` for a 2-dimension array.
6. To define the CMP field as a key field, select the **As key field** checkbox. (If you opened the Create CMP Field SmartGuide from within the Create

© Copyright IBM Corp. 1998, 2000

Enterprise Bean with Inheritance SmartGuide, the **As key field** checkbox does not appear.) The key field icon  will appear next to the name of the CMP field:

In addition, the CMP key field is added to the key class that is associated with the entity bean and a new constructor is created with the new key field as a parameter. Although a new constructor is generated in the key class, the old constructor in the key class is not deleted and a new create method is not generated in the bean class. You either need to modify an existing create() method or create a new create() method.

7. To generate methods to retrieve and set values for the CMP field, select the **Access with getter and setter methods** checkbox.
8. If you have selected the **Access with getter and setter methods** checkbox and you want to promote these generated methods to the remote interface, select the **Promote getter and setter methods to remote interface** checkbox.
9. Select the **Make getter read-only** checkbox if you want to specify the getter method as read-only. This informs the container that the getter method does not update any of the CMP entity bean's CMP fields. The container will not write the bean's data back to the database after the getter method is called, which can improve performance. (Note that this check box is only enabled if you have selected the **Promote getter and setter methods to remote interface** check box.)
10. Do one of the following:

    - If you are generating a getter or setter method, then ensure that the **public** radio buton is selected for the getter or setter method.
    - If you are not generating a getter or setter method, then ensure that the **none** radio button is selected for the getter or setter method.

    Note that if you are editing an existing CMP field, if the field does not have a getter or setter-related method, the **none** radio button is selected by default.
11. If you are using the Create CMP Field SmartGuide and you want to create more than one new CMP field, click the **Create a CMP field and continue** button to immediately create your first new CMP field but have the SmartGuide left open so that you can create another CMP field.
12. Click **Finish**.

If you later decide that you want to change the CMP field and key field definitions for your CMP entity bean, menu items are available that enable you to quickly change the definitions without using the Edit CMP Field SmartGuide. Information about defining CMP field and key field definitions without using the SmartGuide is found in the topic "Defining CMP fields and key fields."

RELATED CONCEPTS

Key classes
Overview of the EJB Development Environment

RELATED TASKS

Using the EJB Development Environment: overview
Defining CMP fields and key fields
Defining database schemas: overview
Mapping database schemas: overview

Defining CMP fields and key fields

There are several ways to add CMP fields to a CMP entity bean and to define one or more of the fields as key fields. You can add CMP fields and specify the key fields when you first create a new CMP entity bean using the Create Enterprise Bean SmartGuide. Or, you can add CMP fields and specify the key fields for an existing CMP entity bean by using the Create CMP Field SmartGuide. You can also edit existing CMP fields by using the Edit CMP Field SmartGuide. Information about creating or editing CMP fields for an existing CMP bean is found in the topic "Adding or editing CMP fields".

However, if want to quickly define (or undefine) existing fields as CMP fields or key fields without using a SmartGuide, you can use the pop-up menu items described in the following sections.

Defining CMP fields

1. In the Enterprise Beans pane of the EJB page, select the entity bean for which you want to define a CMP field.
2. In the upper right corner of the Types pane, click the Fields icon to toggle to the Properties pane.
3. In the Properties pane, select the field that you want to define as container-managed. (To select more than one field, press and hold down the **Ctrl** key while selecting the fields.)
4. Click the right mouse button and select **Container Managed** from the pop-up menu. The following symbol for a container-managed field then appears next to the field:

Removing CMP fields

1. In the Enterprise Beans pane of the EJB page, select the entity bean from which you want to remove a CMP field.
2. In the upper right corner of the Types pane, click the icon to toggle to the Properties pane.
3. In the Properties pane, select the CMP field that you want to remove.
4. Click mouse button 2 and select **Delete** from the pop-up menu. The CMP field and the corresponding getter and setter methods are removed.

Removing the CMP-field designation

1. In the Enterprise Beans pane of the EJB page, select the entity bean that contains the field from which you want to remove the CMP-field designation.
2. In the upper right corner of the Types pane, click the icon to toggle to the Properties pane.
3. Select the CMP field that you want to redefine.
4. Click mouse button 2 and deselect **Container Managed** from the pop-up menu.

   The contained managed icon no longer appears next to the name of the CMP field.

Defining key fields

For an entity bean, you can specify multiple fields that together comprise the key for the class. To designate a field as a key field, do the following:

1. In the Enterprise Beans pane of the EJB page, select the entity bean for which you want to define a key field.
2. In the upper right corner of the Types pane, click the icon to toggle to the Properties pane.
3. In the Properties pane, select the field that you want to define as a key field.
4. Click mouse button 2 and select **Key Field** from the pop-up menu.

   The key field icon appears next to the name of the field. In addition, the key field is added to the key class that is associated with the entity bean and a new constructor is created with the new key field as a parameter. Although a new constructor is generated in the key class, the old constructor in the key class is not deleted, and a new ejbCreate() method is not generated in the bean class. You need to either modify the existing ejbCreate() method or create a new ejbCreate() method.

Removing the key-field designation

1. In the Enterprise Beans pane of the EJB page, select the entity bean that contains the field for which you want to remove a key-field designation.
2. In the upper right corner of the Types pane, click the icon to toggle to the Properties pane.
3. Select the key field that you want to redefine.
4. Click mouse button 2 and deselect **Key field** from the pop-up menu.

   The key field icon no longer appears next to the name of the CMP field. The constructor in the key class with the VA symbol beside it will either be modified or a new one will be generated. Old constructors may also remain in the key class with errors. You need to delete these old constructors. Also, the create() methods in the bean class are not modified. You either need to modify an existing create() method or create a new create() method.

Ensure that there are no problems in your bean and key classes. Select the **All Problems** tab to check for problems. Any problems must be resolved before you can generate deployed code for the EJB group.

RELATED CONCEPTS

Key classes and key fields
Overview of the EJB Development Environment

RELATED TASKS

Using the EJB Development Environment: overview
Adding or Editing CMP fields

Setting the inheritance view

In the Properties pane, you can choose to either display or suppress the display of properties (such as CMP fields, key fields, and roles) that have been inherited by the selected CMP enterprise bean from another CMP enterprise bean.

To set the inheritance view:

1. In the Types pane, click the icon to toggle to the Properties pane.

2. Do one of the following:

   - If you want to display inherited properties, from the **Properties** menu, select **Visibility - All Inherited Properties**.
   - If you want to suppress the display of inherited properties, from the **Properties** menu, select **Visibility - No Inheritance**.

RELATED CONCEPTS

Overview of the EJB Development Environment

RELATED TASKS

Using the EJB Development Environment: overview
Adding enterprise beans with inheritance

Generating a schema and mapping from an EJB group

To employ the top-down approach to mapping enterprise beans to database tables, you can generate a default schema and mapping from an existing EJB group. (Additional information about the top-down approach is found in the topic "Approaches for mapping enterprise beans to database tables.")

If you have previously generated and deleted a schema and map, be sure to delete the corresponding storage classes before regenerating.

To generate a schema and map from an EJB group, follow these steps:

1. In the Enterprise Beans pane of the EJB page, select the EJB group from which you want to generate a default schema and mapping.
2. From the **EJB** menu, select **Add > Schema and Map from EJB Group**.

   The EJB tools create a schema with columns for each CMP field, as well as a map of each CMP field to a column. You can then make any required changes to the schema or map.

   As generated, the lengths of the new schema's element names might be too long for your database. To check this and make adjustments, select **Validate Physical Names** from the **Schemas** menu of the Schema Browser. This modifies the physical name used by the database but leaves the logical name alone.

   If you make changes to the CMP bean and then regenerate the schema and mapping, neither the columns of the original schema nor the property maps of the original map will be changed. If, however, you add new CMP fields before generating the schema and map a second time, additional columns and property maps will be generated for the new CMP fields. If you delete a field or enterprise bean, the corresponding column or table will not be removed.
3. Save the schema and mapping.
4. Export the schema to the database:
   a. In the Schema Browser, select the schema you just created.
   b. From the **Schemas** menu, select **Import/Export Schema** and then **Export Entire Schema to Database**.
   c. Provide the requested information and click **OK.**

You can also export the schema to the database in the EJB Server Configuration Browser; this process is described in the topic "Creating database tables."

If inheritance relationships are involved, VisualAge for Java generates a single-table inheritance map by default.

RELATED CONCEPTS

Overview of the EJB Development Environment
Approaches for mapping enterprise beans to database tables

RELATED TASKS

Using the EJB Development Environment: overview
Creating EJB groups from schemas or models
Defining CMP fields and key fields
Generating EJB deployed classes
Starting the database servers
Creating database tables

RELATED REFERENCES

Type mappings for top-down programming
Default type mappings for deployment to Component Broker

Mapping CMP fields to an existing database table

Defining database schemas

Defining database schemas: overview

Two kinds of enterprise beans can be persistent: entity beans with bean-managed persistence (BMP) and entity beans with container-managed persistence (CMP).

Bean-managed persistence is achieved by writing to a database directly from the entity bean class. For BMP entity beans, you must manually code the persistence support.

However, you can use the Schema and Map Browser tools in the EJB Development Environment to define persistence for your CMP entity beans. Your input through these tools is used to generate persistence-support code.

In the EJB Development Environment, a *schema* represents each CMP entity bean from a database perspective. The schema typically consists of at least one table and its columns. In a schema, you also designate how an object type gets handled when it is stored as a database type (and the opposite). In the case where a field in the enterprise bean corresponds to the column of another table, you can define a foreign-key relationship to reach beyond the scope of the primary table.

Each schema is mapped to a set of physical database tables. The CMP fields are then mapped to columns in database tables.

The bean-to-table mapping is called a *table map*; the field-level mapping is called a *property map*. Both types of maps are explained in more detail in the topic "Mapping database schemas: Overview".

RELATED CONCEPTS

Overview of the EJB Development Environment

RELATED TASKS

Mapping database schemas: overview
Creating schemas
Generating a database schema and mapping from an EJB group
Using the EJB Development Environment: overview

Creating schemas

You can create schemas by defining them manually or by importing them from an existing database.

Creating schemas manually:

1. From the Workbench, open the Schema Browser by clicking the  icon.
2. From the **Schemas** menu, select **New Schema**.
3. Type in a schema name. Use a naming convention style similar to what you would use for naming classes.
4. Click **OK**. The name of the schema appears in the Schemas pane.

   To complete the schema, you must create tables and columns. This is explained in the topic "Creating tables and columns."

Importing schemas from an existing database:

1. From the Workbench, open the Schema Browser by clicking the  icon.
2. From the **Schemas** menu, select **Import/Export Schema** and then **Import Schema from Database**. The Information Required dialog appears.
3. In the **Enter the Name for the Schema** field, type a schema name and click **OK**. Use a naming convention style similar to what you would use for naming classes. The Database Connection Info dialog appears.
4. From the **Connection type** list, select a database driver. The drivers listed in this field are drivers that are currently loaded in your workspace. If you have specified a driver in your class path but it does not display in the list, you can type in the name of a driver. For example,

   ```
   COM.ibm.db2.jdbc.app.DB2Driver
   ```

5. In the **Data source** list, type the name of your data source. This is the name of the database that has been created by you or a database administrator.

   The syntax is jdbc:<subprotocol>:<subname> For example, if you have the native DB2 driver installed, your data source would be as follows:

   ```
   jdbc:db2:sampleDB
   ```

   The format of *subname* is dependent on the kind of subprotocol. Consult your local database administrator if you are not certain of the syntax.
6. Click **OK**. The Select Tables dialog appears.
7. From the **Qualifiers** list, select the qualifier associated with the schema that you want to import.
8. Click the **Build Table List** button.
9. In the Name column, select the tables that you want to import.
10. Click **OK**. The imported database schema appears in the Schemas pane of the Schema Browser.

RELATED CONCEPTS

Overview of the EJB Development Environment

RELATED TASKS

Using the EJB Development Environment: overview
Creating tables and columns
Mapping database schemas: overview

Creating tables and columns

Because entity beans with container-managed persistence (CMP) fields store persistent data in database tables, you must define schema tables and their associated columns. You do this by using the Schema Browser.

This procedure assumes that you have created a schema.

First, create the table:

1. From the Schema Browser, select the schema name in the **Schemas** pane.
2. From the **Tables** menu, select **New Table**. The Table Editor opens.
3. In the **Name** field, type a logical table name. This name will not be stored in the database.
4. In the **Physical name** field, type a physical table name. If you leave this name blank, the physical name will be filled in with what you typed into the **Name** field.

   This name is stored in the database. The length of the physical name may be limited by your database. Consult your database documentation for details.

   If you are doing bottoms-up development and intend to create a default EJB group from the schema, take care in naming. Because these names are used to generate Java class names, make sure that any characters used in the logical name conform to Java class-naming guidelines. In addition, the WebSphere Application Server does not support class or package names that contain double-byte characters.
5. In the **Qualifier** field, type a qualifier name. The length of the qualifier name may be limited by your database. Consult your database documentation for details.

Next, create columns for each of the CMP fields of the enterprise bean that will write in the table. To create a column, do the following:

1. In the **Table columns** list of the Table Editor, click **New**. The Column Editor opens.
2. In the **Name** field, type a column name. This is a logical name only; it is not stored in the database.
3. In the **Physical name** field, type a physical column name. If you leave this name blank, the physical name will be filled in with what you typed into the **Name** field.

   This name is stored in the database. The length of the physical name may be limited by your database. Consult your database documentation for details.

   If you are doing bottoms-up development and intend to create a default EJB group from the schema, take care in naming. Because these names are used to generate Java class names, make sure that any characters used in the logical name conform to Java class-naming guidelines.
4. From the **Type** list, select a column type.
5. From the **Converter** list, select a converter for the column:
   - The default converter, VapConverter, essentially passes the datum as-is without doing any conversion.

- If you need data converted, choose another converter from the list. For a description of each converter shipped with VisualAge for Java, see the contextual help for the Column Editor.

6. If your column must contain a non-null value, make sure that **Allow nulls** is not checked.

   The default for the **Allow nulls** check box is to allow a null value to be stored in the column. Typically, the null value is for columns that contain optional information. Your data design determines if columns can contain the null value.

   Columns that will be defined as primary keys must not allow nulls; that is, a primary key must always have a value. Therefore, for primary key columns, be sure to deselect the **Allow nulls** check box.

7. Click **OK**.

After all of the columns are defined, designate a primary key:

1. From the **Table columns** list, select the names of columns for the primary key.
2. Click the double-arrow (>>) to move each column name into the **Primary key** list.

   It is not recommended that you use a column of type CHAR as your primary key. If you do, specify the converter for that column as VapTrimStringConverter. In addition, you cannot define columns of type BLOB, CLOB, LONG VARBINARY, LONG VARCHAR, or LONG VARGRAPHIC to be primary keys.

When you have defined all of the columns and the primary key in the Table Editor, click **OK**.

The name of the new table is displayed in the Tables pane, and its associated columns are listed in the Columns pane.

RELATED CONCEPTS

Converters
Overview of the EJB Development Environment

RELATED TASKS

Using the EJB Development Environment: overview
Exporting schemas
Mapping database schemas: overview

RELATED REFERENCES

Type mappings for bottom-up programming
Default type mappings for deployment to Component Broker
Data-type converters supported for persistence

Creating foreign-key relationships

You define foreign-key relationships between two schema tables under the following circumstances:

- **Secondary table mapping.** In this case, you relate the primary and secondary tables.
- **Root/leaf inheritance table mapping.** In this case, you relate the table of each leaf with that of its immediate parent in the inheritance tree.
- **Association mapping.** In this case, you relate the two tables that correspond to roles of an association.

Do not designate foreign keys as CMP fields of an enterprise bean if the foreign-key relationship is already mapped to an association in the EJB group. If you do designate the foreign key as a CMP field, you will get an Overlapping Map validation error in the map.

To create a foreign-key relationship, do the following:

1. From the Workbench, open the Schema Browser by clicking the  icon.
2. In the Schemas pane, select the appropriate schema.
3. From the **Foreign_Keys** menu, select **Foreign Key Relationship**. The Foreign Key Relationship Editor opens.
4. In the **Name** field, give the relationship a meaningful name.
5. If a foreign key constraint exists on the database, make sure that **Constraint exists in database** is checked.
6. Edit the contents of the **Relationship** group box:

   - From the **Primary key table** list, select a table. The Primary Key column (read-only) is updated with the primary key from the table.
   - From the **Foreign key table** list, select a table. From the Foreign Key column, select the key.
7. Click **OK**.

RELATED TASKS

Creating table maps
Mapping associations

Writing your own converters

A converter is used to translate a single database column to and from the bean class field. Many converters are available in the VapAbstractConverter class hierarchy. You can also write your own converters.

Note: If you wish to deploy your CMP beans to Component Broker, be aware that usage of VisualAge for Java converters is not directly supported there. However, the Object Builder tool can automatically handle the mapping of some data types in preparation for deployment to Component Broker. For more information, see "Default type mappings for deployment to Component Broker."

To write a converter do the following:

1. Create a new class (for example, MyIntegerToIpAddressConverter) that is a subclass of VapAbstractConverter. The purpose of MyIntegerToIpAddressConverter is to convert a database INTEGER type to a String that represents an IP address.

   Note: Make sure that the class name is absolutely unique; package information is not stored in persistence metadata.
2. Define the following variable in the class definition:

   ```
   static MyIntegerToIpAddressConverter singleton = null;
   ```
3. Implement the following methods:
   - getSourceDatatype() - This method should return the data element's type name. For example:

     ```
     public static String[] getSourceDatatype() {
        String types [] = {"INTEGER"};
        return types;
     }
     ```

- getTargetClassName() - This method returns the target class name. For example:

```
public static String getTargetClassName() {
   return "java.lang.String";
}
```

- objectFrom(Object) - This method returns the object from the database, converted to an instance of the target class.

```
public Object objectFrom (Object aField) {
   // Your converter code goes here and converts
   // aField which is an INTEGER
   // in this example, to a String that represents
   // an IP address("9.37.216.7")

   return ((String) convertedField);
}
```

- dataFrom(Object) - The argument will be an instance of the target class.

```
public Object dataFrom (Object anIpAddressString) {
   // Your converter code goes here which
   // converts anIpAddressString,
   // ("9.37.216.7"), to an INTEGER

   return new java.lang.Integer(aConvertedField);
}
```

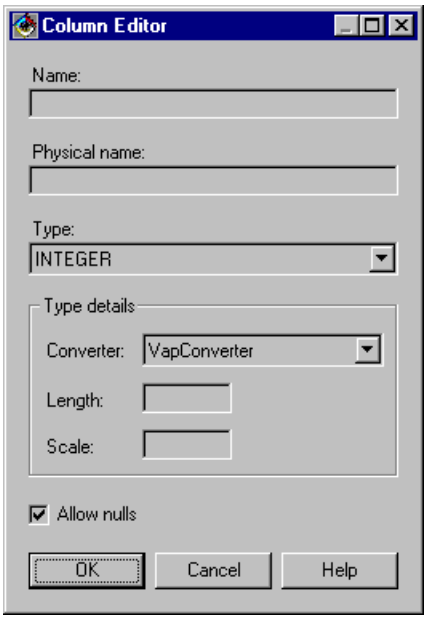
- singleton() - This method returns the sole instance of the converter.

```
public static MyIpAddressConverter singleton ( ) {
   if (singleton = = null )
      singleton = new MyIpAddressConverter();
   return singleton;
}
```

- reset() - Set the singleton variable to null. This is required by each subclass of VapAbstractConverter.

```
public static void reset() {
   singleton = null;
   return;
}
```

4. Use the Column Editor in the Schema Browser to associate the converter to a table column. The converter name that you defined is now selectable from the **Converter** list in the Column Editor.



RELATED CONCEPTS

Overview of the EJB Development Environment
Converters

RELATED TASKS

Using the EJB Development Environment: overview
Creating tables and columns

RELATED REFERENCES

Data-type converters supported for persistence
Default type mappings for deployment to Component Broker

Exporting schemas

Exporting a schema is a quick way to set up tables in your enterprise bean's deployment database. If any of the tables in the schema already exist in the database, this process replaces the tables in the database with those in the schema.

Schema export is not the only way to set up tables. For examples, if you do not have direct access to the deployment server, you can use the "Creating Database Tables" option of the EJB Server Configuration browser. For more information about this process, see the topic "Creating database tables." If this is your situation, you can still use schema export to set up a local test database.

To export an entire schema to the database, follow these steps from the Schema Browser:

1. From the **Schemas** menu, select **Import / Export Schema** and then **Export Entire Schema to Database**.
2. In the **Enter the Name for the Schema** field, type the schema name and click **OK**. The Database Connection Info dialog appears.
3. From the **Connection type** list, select a database driver. The drivers listed in this field are drivers that are currently loaded in your workspace. For example, if you have the DB2 driver loaded in your workspace and class path, the driver would be shown as follows:

```
COM.ibm.db2.jdbc.app.DB2Driver
```

4. In the **Data source** list, type the name of your data source. This is the name of the database that has been created by you or a database administrator.

   The syntax is jdbc:<subprotocol>:<subname>

   For example, if you have the native DB2 driver installed, your data source would be as follows:

   ```
   jdbc:db2:sampleDB
   ```

   The format of the subname is dependent on the kind of subprotocol. Consult your local database administrator if you are not certain of the syntax.
5. Click **OK**. The database schema is exported to the database, and you will receive status messages in the Console window during the export process.

You can also use the **Import/Export Schema** menu to import or export tables and keys selectively.

For information about importing schemas, see the topic "Creating schemas."

RELATED CONCEPTS

Overview of the EJB Development Environment

RELATED TASKS

Using the EJB Development Environment: overview
Creating schemas
Mapping database schemas: overview
Creating database tables

Mapping database schemas

Mapping database schemas: overview

Entity beans with container-managed persistence (CMP) fields must be mapped to database tables that are represented in a schema. You map entity beans to database tables by creating an EJB group map in the Map Browser. EJB group maps are used by VisualAge for Java to generate the SQL and other supporting code needed to make enterprise beans persistent.

Before creating an EJB group map, you must create enterprise beans and a database schema. For more information about schemas, see the topic "Defining database schemas: Overview."

When you generate a default schema for the enterprise beans, a default EJB group map is created. This task is explained in the topic "Generating a schema and mapping from an EJB group."

A default EJB group map is also created when you create an EJB group from a schema. This task is explained in the topic "Creating EJB groups from schemas or models."

The default map is sufficient for most enterprise beans; it is especially useful if you have access only to a local database for unit testing. As needed, you can customize the default map by using the Schema and Map Browser tools.

Typically, there is a one-to-one correspondence between a CMP entity bean and a database table. That is, for each entity bean with CMP fields, there is one database

table. There is also a one-to-one correspondence between each CMP field in the entity bean and a column in the database table.

By default, inheritance structures are mapped as single-table inheritance maps. If you delete all of the table maps within an inheritance map and change the root map to be a root/leaf inheritance map, you can regenerate the default schema and map. The rest of the beans in the inheritance structure are then mapped as a root/leaf inheritance map.

An EJB group map contains a collection of table maps. Each table map contains a collection of property maps. Editing the property maps is how you relate each CMP field in the enterprise bean to a column in the table.

Composers enable one CMP field to be mapped to multiple columns. You assign a composer class to the field; when you create a default schema, the corresponding composer is automatically used to map the CMP field to the correct columns in the newly created table. For example, a CMP field of type VapName might use VapNameComposer to map the first-, middle-, and last-name properties of VapName to FIRST, MIDDLE, and LAST columns in a database table.

When all of the table maps are defined, the persistence code for your enterprise beans can be generated from the **EJB** menu of the Workbench.

RELATED CONCEPTS

Overview of the EJB Development Environment

RELATED TASKS

Generating a schema and mapping from an EJB group
Creating EJB groups from schemas or models
Creating EJB group maps
Defining database schemas: overview
Generating EJB deployed classes
Using the EJB Development Environment: overview

RELATED REFERENCES

Type mappings for bottom-up programming
Type mappings for top-down programming

Creating EJB group maps

For every EJB group that has entity beans with container-managed persistence fields, an EJB group map must be created. To create an EJB group map, do the following:

1. Open the Map Browser by clicking the  icon.
2. From the **Datastore_Maps** menu, select **New EJB Group Map**. The New Datastore Map dialog appears.
3. Type the name of the EJB group map in the **Name** field. Use the same naming convention as you would when naming a class.
4. From the drop-down lists, select an EJB group name and database schema name.
5. Click **OK**.

The new EJB group map is created and its name is displayed in the Datastore Maps pane. The following "mapped" icon appears in the Enterprise Beans pane beside each mapped enterprise bean: 

Next, you continue to define your EJB group map by defining table and property maps. When all of the table maps and property maps are defined, you can generate the deployment code for the enterprise beans.

RELATED CONCEPTS

Overview of the EJB Development Environment

RELATED TASKS

Using the EJB Development Environment: overview
Defining database schemas: overview
Creating table maps
Editing property maps
Generating EJB deployed classes

Creating table maps

When you select an EJB group map in the Map Browser, entity beans with container-managed persistence (CMP) fields are displayed in the Persistent Classes pane. Each entity bean with CMP fields must have at least one table map. The table map specifies the database table to which the entity bean is mapped.

Secondary table maps enable the mapping of CMP fields to more than one table. For example, an enterprise bean named Department could have CMP fields like *name*, *courses*, and *staff* mapped to the primary table (DEPT) and have other CMP fields such as *phoneNumber* mapped to a secondary table (DEPT_EXTRAS).

Note: Mapping an enterprise bean to multiple data stores is not supported.

A foreign-key relationship must be defined between the primary and secondary tables before you can define the secondary table map. This foreign-key relationship must be a primary-key to primary-key relationship. For more information about this task, see "Creating foreign-key relationships."

If your enterprise beans use inheritance, you have the following mapping options:

- You can combine all CMP fields for an inheritance hierarchy in one table map. A single-table inheritance table-map is used to map two or more enterprise beans to the same table. This is useful for enterprise beans that have basically the same data but different behaviors.
- You can use a table map for each enterprise bean and establish the necessary foreign key relationships between the tables. These foreign-key relationships must be primary-key to primary-key relationships. This *root/leaf*-inheritance table map is similar to the single-table inheritance table-map, except that each enterprise bean can have its own mapping, stored in a separate table map. The enterprise bean obtains information from its table map as well as from its superclasses' table maps. Foreign-key relationships must be defined between the root table and all leaf tables before you can define this type of mapping.

Creating a single table map with no inheritance

1. In the Persistent Classes pane, select the enterprise bean for which you want to create a table map.

2. From the **Table_Maps** menu, select **New Table Map** and then **Add Table Map with No Inheritance.**
3. From the **Table** list, select the name of the schema table to be mapped.
4. Click **OK**. The name of the table map is displayed in the Table Maps pane of the Map Browser.
5. Map the CMP fields and associations for the table map. For more information about this process, see the topics "Editing property maps" and "Mapping associations."

Creating primary and secondary table maps

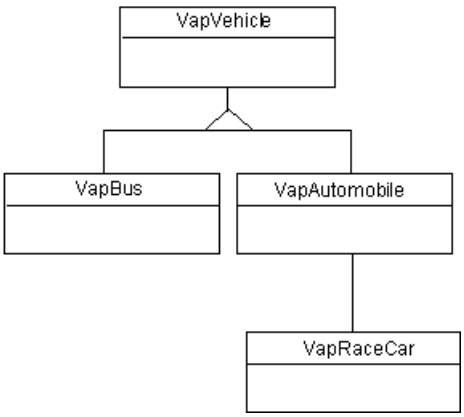
1. As mentioned previously, define a foreign-key relationship between the two tables you wish to map. This must be a primary-key to primary-key foreign-key relationship.
2. In the Persistent Classes pane, select the enterprise bean that contains the primary-key table you specified in step 1.
3. From the **Table_Maps** menu, select **New Table Map** and then **Add Table Map with No Inheritance.**
4. From the **Table** list, select the name of the primary-key table specified in the foreign-key relationship that you defined in step 1.
5. Click **OK**.
6. Map the CMP fields and associations for the primary table map.
7. From the **Table_Maps** menu, select **New Table Map** and then **Add Secondary Table Map**.
8. From the **Table** list, select the name of the foreign-key table specified in the foreign-key relationship that you defined in step 1.
9. Map the CMP fields and associations for the secondary table map.

Creating a single table map for an inheritance hierarchy

1. In the Persistent Classes pane, select the first enterprise bean for which you want to create a table map.
2. From the **Table_Maps** menu, select **New Table Map** and then **Add Single Table Inheritance Table Map**.
3. From the **Table** list, select the name of the schema table to be mapped.
4. From the **Discriminator column** list, select the name of the column to be mapped.
5. In the **Discriminator value** field, type a value that will uniquely identify this enterprise bean in the table map. The value is validated against the corresponding schema type.
6. If this enterprise bean is the root of the hierarchy, make sure that **Root of the hierarchy** is checked.
7. Map the CMP fields and associations for the enterprise bean.
8. Repeat these steps for each enterprise bean in the hierarchy, choosing a different discriminator value for each one.

Creating root/leaf-inheritance table maps

Consider the following inheritance tree as you read through this process:



1. As mentioned previously, define foreign-key relationships between the tables you wish to map. These must be primary-key to primary-key foreign-key relationships.

   For enterprise beans in the tree shown previously, the following foreign-key relationships exist:

   - bus to vehicle
   - automobile to vehicle
   - racecar to automobile
2. In the Persistent Classes pane, select the enterprise bean for which you want to create a table map.
3. From the **Table_Maps** menu, select **New Table Map** and then **Add Root/Leaf Inheritance Table Map**.
4. From the **Table** list, select the name of the schema table to be mapped.
5. If this enterprise bean is the *root* of the inheritance hierarchy, do the following:
   - Select a column name from the **Discriminator column** list.
   - Make sure that **Root of the hierarchy** is checked.
   - Click **OK**.
6. If this enterprise bean is not the root of the inheritance hierarchy, do the following:
   - In the **Discriminator value** field, type a value that will uniquely identify this enterprise bean in the table map.
   - In the **Foreign key relationship** list, select or enter the appropriate value based on the relationship of this enterprise bean to the rest of the tree.
   - Click **OK**.
7. Map the CMP fields and associations for the table map.
8. Starting with step 2, repeat this process for each enterprise bean in the hierarchy, choosing a different discriminator value and foreign-key relationship for each one.

RELATED CONCEPTS

Overview of the EJB Development Environment

RELATED TASKS

Creating foreign-key relationships
Editing property maps
Mapping associations

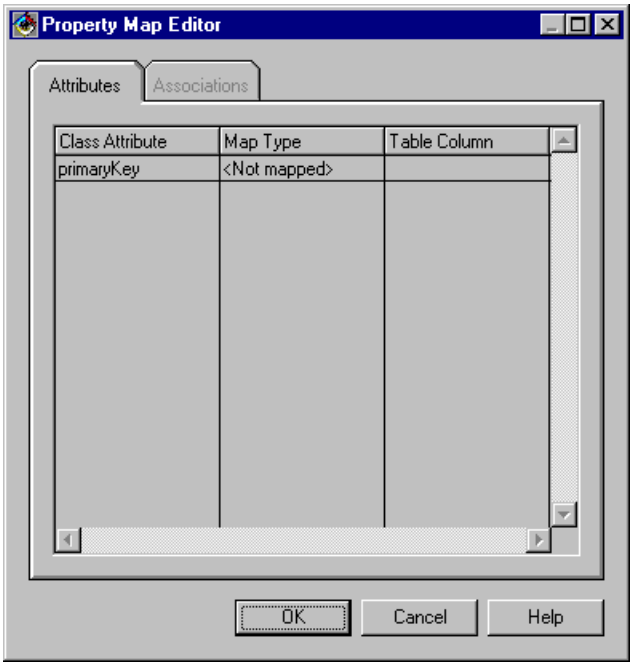
Defining database schemas: overview
Using the EJB Development Environment: overview
Generating EJB deployed classes

Editing property maps

When it is first created, each table map in an EJB group map contains a collection of property maps. A property map exists for each container-managed (CMP) field that you defined in your enterprise bean. When you edit a property map, you associate the CMP fields in the enterprise bean with the columns of a database table.

To edit a collection of property maps, do the following:

1. In the Map Browser, select an enterprise bean from the Persistent Classes pane and a table map from the Table Maps pane.
2. From the **Table_Maps** menu, select **Edit Property Maps**. The Property Map Editor opens.



3. On the Attributes page, map the CMP fields to table columns as follows:

   a. Select an entry from the Class Attribute column.
   b. Click in the Map Type column to choose a map type for the attribute.

      - If you want to map one attribute to one column in the database table, choose **Simple**.
      - If the field you are mapping is a first-class object that consists of other fields and behavior, choose **Complex**. You must also create a composer for it. Refer to the topic "Mapping complex attributes using composers" for further details.

   c. Click in the Table Column field to choose the name of a column to relate to the attribute.
4. Click **OK**.

RELATED CONCEPTS

RELATED TASKS

Mapping complex attributes using composers
Mapping associations
Using the EJB Development Environment: overview
Defining database schemas: overview
Generating EJB deployed classes

Mapping complex attributes using composers

A composer is used to map a single complex bean field to multiple database columns. Composition is needed for complex fields that are themselves objects with fields and behavior. As an example, this topic explains how to create a composer for a complex field named *customerName*. This example is based on *com.ibm.vap.BankSample.VapNameComposer*, a Persistence Builder sample.

Suppose you have a VapCustomer bean that has a complex field called *customerName*. This *customerName* field is stored in a database table called CUSTOMER. The three column names are FIRSTNAME, MIDINIT, and LASTNAME.

In order to map *customerName*, you must create a composer class that must be subclassed from VapAttributeComposer. Once a composer is created, you can use it in the Map Browser.

To create a composer, do the following:

1. Create a new class that extends VapAttributeComposer. For this example, name the class VapCustomerNameComposer.

   Note: Make sure that the class name is absolutely unique; package information is not stored in persistence metadata.
2. Define the following variable in the class definition:

   ```
   private static VapCustomerNameComposer singleton;
   ```
3. Implement the following methods:
   - getSourceDatatype() - This method returns an array of the data elements' class names passed as a parameter to the objectFrom(Object) message. For example:

     ```
     public static String[] getSourceDatatype() {
         String[] types = { "String", "String", "String" };
         return types;
     }
     ```
   - getTargetClassName() - This method returns the name of the class for the instance created as a result of the objectFrom(Object) method. The target class in this scenario would be VapCustomerName. For example:

     ```
     public static String getTargetClassName() {
        return "yourPackageName.VapCustomerName";
     }
     ```
   - getAttributeNames() - This method returns an array of the attribute names from the target class. For example:

     ```
     public static String[] getAttributeNames() {
        String[] attributes = { "FirstName", "Middle", "LastName" };
        return attributes;
     }
     ```
   - objectFrom(Object) - The argument is an array that contains the values for the attribute string name from the method getAttributeNames() in the same

order. This method sets the target class attributes based on these values and returns an instance of the target class, as follows:

```
public Object objectFrom (Object[] anArray) {
   String first, middle, last;
   first = (String) anArray[0];
   middle = (String) anArray[1];
   last = (String) anArray[2];
   return new VapCustomerName(first, middle, last);
}
```

- dataFrom(Object) - This method is responsible for returning a collection of objects that are to map to data store fields. The argument is an instance of the target class. For example:

```
public Object[] dataFrom (Object anObject) {
   VapName name = (VapName) anObject;
   Object[] anArray = new Object[3];
   if (anObject == null) {
      Object[] anArray2 = { null, null, null };
      return anArray2;
   } else {
      anArray[0] = name.getFirstName();
      anArray[1] = name.getMiddle();
      anArray[2] = name.getLastName();
   }
   return anArray;
}
```

- singleton() - This method returns the sole instance of the composer.

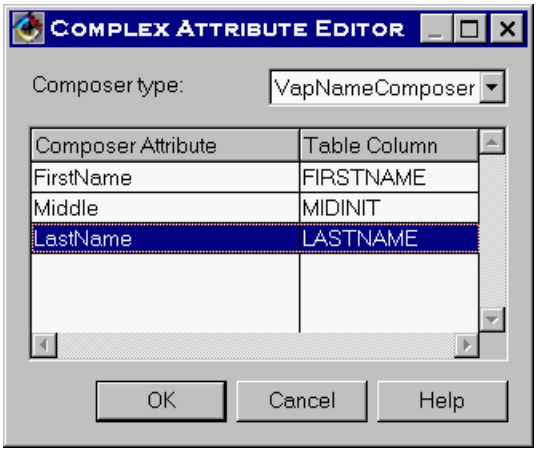
```
public static VapCustomerNameComposer singleton() {
   if (singleton == null)
      singleton = new VapCustomerNameComposer();
   return singleton;
}
```

- reset() - This method is required by each subclass of VapAttributeComposer. Set the singleton variable to null. For example:

```
public static void reset() {
   singleton = null;
   return;
}
```

4. After you have created the class and implemented the above methods, use the Map Browser to map the complex field *customerName* to the three columns in the database table CUSTOMER.

   Do this from the Property Map Editor. When you map the field, you must select **Complex** from the **Map Type** list; then click in the field **Table Column** to expose a small button. Click the button; the Complex Attribute Editor appears.



Select the new composer from the **Composer type** list. The mapped attributes should then appear in the table underneath.

RELATED CONCEPTS
Overview of the EJB Development Environment

RELATED TASKS
Editing property maps
Using the EJB Development Environment: overview

Creating or editing associations

Note: Support for associations has been included in VisualAge for Java as a technical preview. It can be used for deployment to the WebSphere Application Server, Advanced Edition, or for EJB JAR exportation to Component Broker (WebSphere Application Server, Enterprise Edition). Limitations exist for Component Broker deployment; see the Component Broker documentation for details.

Creating associations

To create an association between enterprise beans, do the following:

1. In the Workbench, select the EJB tab.
2. Select the appropriate EJB group; then click the right mouse button.
3. From the **EJB** menu, select **Add - Association**.
4. In the **Association name** field, type a meaningful name for the association.
5. In the left **Enterprise bean** group, select one enterprise bean to be associated.
6. In the right **Enterprise bean** group, select the other enterprise bean to be associated.
7. Give each association role a name.
8. Set parameters for each role, as follows:
   - **Navigable** means that instances of the other bean in the association can be retrieved using this role.
   - **Many** means that the role potentially represents many instances of the other bean.
   - **Required** means that the bean *must* hold a reference to the other bean. (The reference cannot be null.)
   - **Foreign Key** means that the specified end of the association holds a foreign key for the other end.

     For more information about these settings, see F1 help for the Association Editor.
9. Click **OK**.

After the association has been created, it must be mapped. For more information, see the topic “Mapping associations.”

If you intend to access the enterprise beans from a Java application, you will need to add the IBM EJB Tools project to the class path of the application. For example, if you are using associations and call an association method on the remote interface that returns an enumeration of enterprise beans, the method call will throw an exception unless this project is in the class path. The **Compute now** button for the class path on a class’ properties dialog will not usually add the project; it must be added manually.

Editing or deleting associations

To edit or delete an association, do the following:

1. In the Properties pane of the Workbench, select the appropriate association or role. (Regardless of whether you select association or role, you will operate on the entire association.)
2. From the association's pop-up menu, select one of the following:
   - **Edit** to open the Association Editor
   - **Delete** to delete the association

RELATED CONCEPTS

Inheritance and association

RELATED TASKS

Creating associations by example
Mapping associations
Adding or removing association roles in key classes

RELATED REFERENCES

Finder helpers generated for deployment to Component Broker

Creating associations by example

Note: Support for associations has been included in VisualAge for Java as a technical preview. It can be used for deployment to the WebSphere Application Server, Advanced Edition, or for EJB JAR exportation to Component Broker (WebSphere Application Server, Enterprise Edition). Limitations exist for Component Broker deployment; see the Component Broker documentation for details.

The following examples show how to construct a one-to-one (1:1) or one-to-many (1:M) relationship within an association. To illustrate the different types, we use relationships you might find at a bank or in a group of banks, as follows:

- Between a customer and the customer's address (1:1)
- Between a bank branch and the accounts held within the branch (1:M)

Creating one-to-one relationships

This example explains how to create a 1:1 relationship between Customer and Address beans.

1. Open the Association Editor.
2. In the **Association name** field, type `Customer-BillingAddress`
3. In the left **Enterprise bean** group box, select **Customer**.
4. In the right **Enterprise bean** group box, select **Address**.
5. For Customer, define the association role of Address:
   a. For **Role of Address**, type `billingAddress`
   b. Select **Navigable**.
6. For Address, define the association role of Customer:
   a. For **Role of Customer**, type `customerForBillingAddress`
   b. Select **Navigable**.
   c. Select **Foreign Key**.

7. Click **OK**.

As a result, the following items are generated in the Address enterprise bean:

1. In the remote interface, declarations for accessor methods:
   - getCustomerForBillingAddress()
   - getCustomerForBillingAddressKey()
   - privateSetCustomerForBillingAddressKey(CustomerKey)
   - secondarySetCustomerForBillingAddress(Customer)
   - setCustomerForBillingAddress(Customer)

   private- and *secondary-* methods are for internal association maintenance only.
2. In the bean class, both fields and methods:
   - customerForBillingAddress_customerNumber, a field of type String. This is a copy of the foreign key from Customer.
   - customerForBillingAddressLink, a field to hold a general-purpose link-management object.
   - getCustomerForBillingAddressLink(), an accessor method.
   - _getLinks(), a method to retrieve association links for the bean. This method is generated at bean creation time and then updated with each new or migrated association.
   - _initLinks(), a method to initialize association links for the bean. This method is called in the ejbActivate(), ejbCreate(), and ejbLoad() methods. This method is generated at bean creation time and then updated with each new or migrated association.
   - _removeLinks().
   - Definitions for the methods listed in item 1.
3. In the home interface, a declaration for findBillingAddressByCustomerForBillingAddress(CustomerKey), a finder method.
4. AddressToCustomerForBillingAddressLink, a class that manages this end of the association.

The following items are generated in the Customer enterprise bean:

1. In the remote interface, declarations for accessor methods:
   - getBillingAddress()
   - secondarySetBillingAddress(Address)
   - setBillingAddress(Address)

   The *secondary-* method is for internal association maintenance only.
2. In the bean class, both fields and methods:
   - billingAddressLink, a field to hold a link-management object.
   - getBillingAddressLink(), an accessor method that creates the link-management object if it does not already exist.
   - _getLinks().
   - _initLinks().
   - Definitions for the methods listed in item 1.
3. CustomerToBillingAddressLink, a class that manages this end of the association.

Creating one-to-many relationships

This example explains how create a 1:M relationship between a BankBranch bean and multiple instances of an Account bean.

1. Open the Association Editor.
2. In the **Association name** field, type `BankBranch-Account`
3. In the left **Enterprise bean** group box, select **Account**.
4. In the right **Enterprise bean** group box, select **BankBranch**.
5. Define the association role of BankBranch within Account:
   a. For **Role of BankBranch**, type `branch`
   b. Select **Navigable**.
   c. Select **Foreign Key**.
6. Define the association role of Account within BankBranch:
   a. For **Role of Account**, type `accounts`
   b. Select **Navigable**.
   c. Select **Many**.
7. Click **OK**.

RELATED CONCEPTS

Inheritance and association

RELATED TASKS

Creating or editing associations
Mapping associations by example

Mapping associations

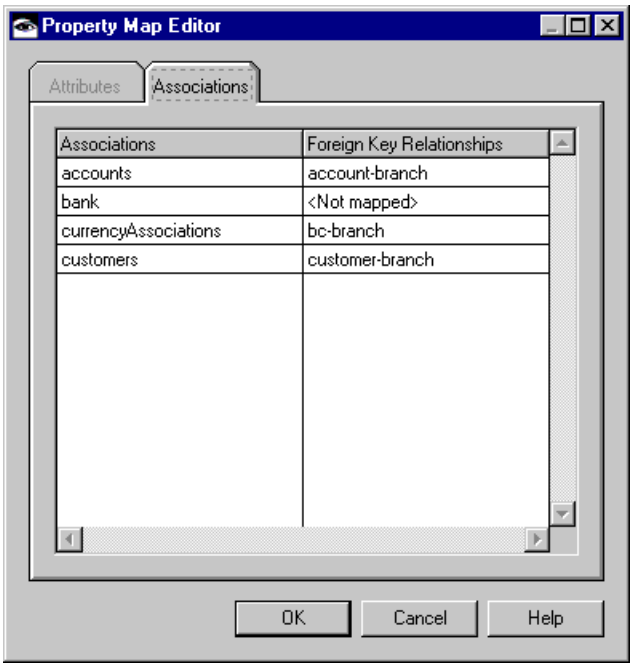
Note: Support for associations has been included in VisualAge for Java as a technical preview. It can be used for deployment to the WebSphere Application Server, Advanced Edition, or for EJB JAR exportation to Component Broker (WebSphere Application Server, Enterprise Edition). Limitations exist for Component Broker deployment; see the Component Broker documentation for details.

After you have created an association between two enterprise beans, you must map them to foreign-key relationships. This process consists of the following:

- In the schema, create the foreign-key relationship that corresponds to the association. For more information about this task, see the topic "Creating foreign-key relationships."
- Map each association role to the foreign-key relationship.

To map association roles to a foreign-key relationship, do the following:

1. From the Workbench, open the Map Browser by clicking the  icon.
2. In the Persistent Classes pane, select one of the associated enterprise beans.
3. In the Table Maps pane, select the table that corresponds to one of the association roles.
4. From the **Table_Maps** menu, select **Edit Property Maps**. The Property Map Editor opens.
5. Select the **Associations** tab.



6. Find the role in the **Associations** column of the table.
7. In the **Foreign-Key Relationships** column, change the **<Not mapped>** designation to the name of the foreign-key relationship you defined earlier.
8. Click **OK**.
9. Repeat steps 2 through 8 for the other association role.

RELATED CONCEPTS

Inheritance and association
Overview of the EJB Development Environment

RELATED TASKS

Creating foreign-key relationships
Creating or editing associations
Mapping associations by example
Using the EJB Development Environment: overview

Mapping associations by example

Note: Support for associations has been included in VisualAge for Java as a technical preview. It can be used for deployment to the WebSphere Application Server, Advanced Edition, or for EJB JAR exportation to Component Broker (WebSphere Application Server, Enterprise Edition). Limitations exist for Component Broker deployment; see the Component Broker documentation for details.

Mapping an association involves two steps:

- Creating a foreign-key relationship that corresponds to the association
- Mapping each association role to the (same) foreign-key relationship

For general information about mapping, see the topic "Mapping associations."

This example continues from the topic "Creating associations by example." It is based on a banking sample that is shipped as com.ibm.vap.BankSample in the

VisualAge Persistence Examples project. For information about creating EJB groups from Persistence Builder models, see the topic “Creating EJB groups from schemas or models.”

Mapping one-to-one relationships

Open the Schema Browser, select **Bank Sample** from the Schemas pane, and select **CUSTOMER** from the Tables pane.

The **customer-billingaddress** foreign-key relationship corresponds to the Customer-BillingAddress association. Create the foreign-key relationship as follows:

1. From the **Foreign_Keys** menu, select **Foreign Key Relationship**. The Foreign Key Relationship Editor opens.
2. In the **Name** field, type `customer-billingaddress`
3. Update the contents of the **Relationship** group box:

    a. Select **ADDRESS** as the primary-key table. The **Primary key** column (read-only) will be updated with the primary key from the table.
    b. Select **CUSTOMER** as the foreign-key table. From the **Foreign key** column, select the foreign key in the CUSTOMER table that corresponds to the billing address, **BILLADDR**.
4. Click **OK**.

Next, map each association role to the foreign-key relationship that you just created.

The **(r) billingAddress(customer-billingAddress)** property map relates the billingAddress association role from Customer to the customer-billingAddress foreign-key relationship.

Map the billingAddress role to the foreign-key relationship as follows:

1. Open the Map Browser. Select **Bank Sample** from the Datastore Maps pane, select **Customer** from the Persistent Classes pane, and select **CUSTOMER** from the Table Maps pane.
2. From the **Table_Maps** menu, select **Edit Property Maps**. The Property Map Editor opens.
3. Click the **Associations** tab.
4. In the table next to **billingAddress**, change **<Not Mapped>** to **customer-billingaddress**.
5. Click **OK**.

In the Address bean, map the customerForBillingAddress role to the same foreign-key relationship as follows:

1. In the Map Browser, select **Address** from the Persistent Classes pane, and select **ADDRESS** from the Table Maps pane.
2. From the **Table_Maps** menu, select **Edit Property Maps**. The Property Map Editor opens.
3. Click the **Associations** tab.
4. In the table next to **customerForBillingAddress**, change **<Not Mapped>** to **customer-billingaddress**.
5. Click **OK**.

Mapping one-to-many relationships

Open the Schema Browser, select **Bank Sample** from the Schemas pane, and select **BRANCH** from the Tables pane.

The **account-branch** foreign-key relationship corresponds to the BankBranch-Account association. Create the foreign-key relationship as follows:

1. From the **Foreign_Keys** menu, select **Foreign Key Relationship**. The Foreign Key Relationship Editor opens.
2. In the **Name** field, type `account-branch`
3. Update the contents of the **Relationship** group box:

    a. Select **BRANCH** as the primary-key table. The **Primary key** column (read-only) will be updated with the primary key from the table.
    b. Select **ACCOUNT** as the foreign-key table. From the **Foreign key** column, select the foreign key in the ACCOUNT table that corresponds to the branch, **BRANCHNO**.
4. Click **OK**.

Next, map each association role to the foreign-key relationship that you just created.

The **(r) accounts(account-branch)** property map relates the accounts association role from BankBranch to the account-branch foreign-key relationship.

Map the accounts role to the foreign-key relationship as follows:

1. Open the Map Browser. Select **Bank Sample** from the Datastore Maps pane, select **BankBranch** from the Persistent Classes pane, and select **BRANCH** from the Table Maps pane.
2. From the **Table_Maps** menu, select **Edit Property Maps**. The Property Map Editor opens.
3. Click the **Associations** tab.
4. In the table next to **accounts**, change **<Not Mapped>** to **account-branch**.
5. Click **OK**.

In the Account bean, map the branch role to the same foreign-key relationship as follows:

1. In the Map Browser, select **Account** from the Persistent Classes pane, and select **ACCOUNT** from the Table Maps pane.
2. In the table next to **branch**, change **<Not Mapped>** to **account-branch**.
3. Click **OK**.

Mapping many-to-many relationships

Open the Schema Browser, select **Bank Sample** from the Schemas pane, and select **BRNCHCURR** from the Tables pane.

You might recall from the topic “Creating associations by example” that the BranchToCurrency object exists only as an intermediary for representing this many-to-many relationship in two one-to-many relationships. The table for BranchToCurrency, BRNCHCURR, contains only two columns:

- BRANCH, a foreign key from the BRANCH table of BankBranch
- TYPE, a foreign key from the CURRENCY table of Currency

The resulting **bc-branch** and **bc-currency** foreign-key relationships correspond to the bc-to-branch and bc-to-currency associations. Create these foreign-key relationships as follows:

1. Create the bc-branch relationship: From the **Foreign_Keys** menu, select **Foreign Key Relationship**. The Foreign Key Relationship Editor opens.
2. In the **Name** field, type `bc-branch`
3. Update the contents of the **Relationship** group box:

    a. Select **BRANCH** as the primary-key table. The **Primary key** column (read-only) will be updated with the primary key from the table.
    b. Select **BRNCHCURR** as the foreign-key table. In the **Foreign key** column, select the foreign key in the BRNCHCURR table that corresponds to the branch, **BRANCH**.
4. Click **OK**.
5. Create the bc-currency relationship: From the **Foreign_Keys** menu, select **Foreign Key Relationship**. The Foreign Key Relationship Editor opens.
6. In the **Name** field, type `bc-currency`
7. Update the contents of the **Relationship** group box:

    a. Select **CURRENCY** as the primary-key table. The **Primary key** column (read-only) will be updated with the primary key from the table.
    b. Select **BRNCHCURR** as the foreign-key table. In the **Foreign key** column, select the foreign key in the BRNCHCURR table which corresponds to the currency, **TYPE**.
8. Click **OK**.

Next, map each association role to the foreign-key relationships that you just created.

BranchToCurrency plays a different role in each of the two foreign-key relationships. For example, the **(r) branch (bc-branch)** property map relates the currency association role from BranchToCurrency to the bc-currency foreign-key relationship.

Map the BranchToCurrency roles to corresponding foreign-key relationships as follows:

1. Open the Map Browser. Select **Bank Sample** from the Datastore Maps pane, select **BranchToCurrency** from the Persistent Classes pane, and select **BRNCHCURR** from the Table Maps pane.
2. From the **Table_Maps** menu, select **Edit Property Maps**. The Property Map Editor opens.
3. Click the **Associations** tab.
4. In the table next to **branch**, change **<Not Mapped>** to **bc-branch**.
5. In the table next to **currency**, change **<Not Mapped>** to **bc-currency**.
6. Click **OK**.

Next, map the other association role (currencyAssociations) for the bc-branch foreign-key relationship, as follows:

1. In the Map Browser, select **BankBranch** from the Persistent Classes pane, and select **BRANCH** from the Table Maps pane.

2. From the **Table_Maps** menu, select **Edit Property Maps**. The Property Map Editor opens.
3. Click the **Associations** tab.
4. In the table next to **currencyAssociations**, change **<Not Mapped>** to **bc-branch**.
5. Click **OK**.

Next, map the other association role (branchAssociations) for the **bc-currency** foreign-key relationship, as follows:

1. In the Map Browser, select **Currency** from the Persistent Classes pane, and select **CURRENCY** from the Table Maps pane.
2. From the **Table_Maps** menu, select **Edit Property Maps**. The Property Map Editor opens.
3. Click the **Associations** tab.
4. In the table next to **branchAssociations**, change **<Not Mapped>** to **bc-currency**.
5. Click **OK**.

RELATED CONCEPTS

Inheritance and association

RELATED TASKS

Mapping associations
Creating associations by example
Creating EJB groups from schemas or models

Adding or removing association roles in keys

Note: Support for associations has been included in VisualAge for Java as a technical preview. It is meant for deployment only with WebSphere Application Server, Advanced Edition.

The association role to be added must have the following characteristics:

- It is not many-valued.
- It is both required and navigable.
- It holds a foreign key.

For more information on these terms, see the F1 help for the Association Editor.

Adding roles to keys

The following task refers to two enterprise beans: one that contains a role (*this* bean) and one represented in the role (the *other* bean).

1. In the Enterprise Beans pane, select the enterprise bean that contains the association role that you want to add to the key.
2. In the Types pane, click the icon to toggle to the Properties pane.
3. In the Properties pane, select the role that you want to add to the key class.
4. From the **Properties** menu, select **Add role to key**.

   This action adds the key fields of the other bean to both the key class and the bean class of this bean. When a role is added to the primary key, VisualAge for Java also regenerates the association.

Be careful when removing beans from an association. If you have added one of the roles to the key of one of the beans, the bean with the role in the key must be removed before the other bean. A database constraint may enforce this (even if the role is not part of the key). However, if there is no database constraint, problems could still occur. For example, suppose you have Customer and Address beans in a 1:1 association where the Address bean's customer role holds the foreign key and that the customer role is the Address bean's primary key. If you were to remove the Customer bean without first removing the Address bean, the association maintenance code, while trying to set the Address bean's customer role foreign-key to null, will also be trying to set the Address bean's primary key to null.

Removing roles from key classes

1. In the Enterprise Beans pane, select the enterprise bean that contains the association role that you want to remove.
2. In the Types pane, click the  icon to toggle to the Properties pane.
3. In the Properties pane, select the role that you want to remove from the key.
4. From the **Properties** menu, select **Remove role from key**.

   When a role is removed from the primary key, VisualAge for Java also regenerates the association.

RELATED CONCEPTS

Key classes and key fields
Overview of the EJB Development Environment
Inheritance and association

RELATED TASKS

Using the EJB Development Environment: overview
Creating or editing associations

Chapter 6. Setting descriptor properties and generating deployed classes

Setting EJB deployment and control descriptor properties

In the following two sections, information is provided on setting deployment descriptor and control descriptor properties for your enterprise beans.

Setting EJB deployment descriptor properties

To set deployment descriptor properties for an enterprise bean:

1. In the Enterprise Beans pane of the EJB page, select the enterprise bean for which you want to set the properties.
2. Click mouse button 2 and select **Properties** from the pop-up menu. The Bean page of the Properties dialog appears. (The Bean page for a session bean is different than it is for an entity bean.)
3. If you need to change the Java Naming and Directory Interface (JNDI) name or other attributes at the bean level, do one or more of the following:

   - In the **Enter the JNDI name for BeanHome** field, type the JNDI name to associate with the enterprise bean in the JNDI name space. The container will bind the enterprise bean's home interface with a JNDI name that includes the name you specified.
   - In the **Transaction Attribute** drop-down box, select a transaction attribute. This attribute tells the container how to manage transaction scopes before and after the execution of the enterprise bean method.
   - In the **Isolation Level** drop-down box, select an isolation level. This level tells the container what isolation level to set on the database connections used by the enterprise bean at the start of each transaction.
   - In the **Run-As Mode** drop-down box, select a run-as mode. This selection tells the container the security identity to associate with the execution of the enterprise bean method.
   - For session enterprise beans only, in the **State Management Attribute** drop-down box, specify whether or not you want the container to maintain state information for the enterprise bean. (Note that if you generate deployed code for a session bean and then later change the state management attribute, you will need to regenerate the deployed code. Note also that session beans that reside in an inheritance hirachy must all have the same value specified for the state management attribute. You can only change this attribute at the parent session bean. Once changed, the attributes for all children session beans will be changed.)
   - For session beans only, in the **Session Timeout Value (seconds)** field, specify how many seconds will elapse before a session times out.
   - For entity beans only, if the enterprise bean is re-entrant, select the **Re-entrant** check box.
4. If you need to change environment properties, click the Environment tab. The Environment page appears. Do one or more of the following:

   - To select an environment property defined for the enterprise bean, click on the properties in the **Environment Properties** list, and click **OK**.

© Copyright IBM Corp. 1998, 2000

- To set an environment property, specify the variable in the **Variables** field and its associated value in the **Values** field, and click **Set**. The property appears in the **Environment Properties** list.
- To delete an environment property, select the property you want to delete and click **Delete**.

Setting EJB control descriptor properties

To set a method-level control descriptor property:

1. In the Enterprise Beans pane of the EJB page, select the enterprise bean for which you want to set the properties.
2. Do one or more of the following:

   - If you want to add a control descriptor for a method, select the method in the Members pane, then select **Members - EJB Method Attributes - Add Control Descriptor** to open the Add Control Descriptor dialog. Accept the default values for the descriptors or change them as necessary, then click **OK**.
   - If you want to edit a control descriptor for a method, select the method in the Members pane, then select **Members - EJB Method Attributes - Edit Control Descriptor** to open the Edit Control Descriptor dialog. Change the descriptors as necessary, then click **OK**.
   - If you want to remove a control descriptor for a method, select the method in the Members pane, then select **Members - EJB Method Attributes - Delete Control Descriptor**.
3. If you want to define a method as read-only, select the method in the Members pane, then select **Members - EJB Method Attributes - Read-only Method.** This toggle menu choice defines and marks the selected method as read-only. This tells the container that the method does not update any of the fields for the BMP or CMP entity bean. The container will not write the bean's data back to the database after the method is called, which can improve performance.

Note that if you are using EJB inheritance and you override a method in the bean class that is marked read-only, the overriding method in the Members pane will display the read-only icon. This is because it is inheriting the read-only attribute from the parent bean. The overriding method will be read-only when it is executed. However, in the pop-up menu for the method, the menu item will indicate that the method is not marked read-only. This is because there is no read-only attribute on the method itself. In effect, there are three states that an overriding method can have with respect to read-only: on, off, and inherit-from-parent.

Similarly, if you override a method in the bean class that has a control descriptor, the overriding method in the Members pane will display the control descriptor icon because it is inheriting the control descriptor from the parent bean. Again, in the pop-up menu for the method, the **Add Control Descriptor** menu item will be active because the overriding method itself does not have its own control descriptor.

For the read-only attribute, if you explicitly set it on for an overriding method, it will no longer be inherited from the parent, even if you later set it to off. The only way to get it back to the inherit-from-parent state is to remove the method from the remote interface and then add it back. For the control descriptor, if you explicitly add one to an overriding method, you can remove it with the **Delete Control Descriptor** menu item from the pop-up menu.

RELATED CONCEPTS

Overview of the EJB Development Environment

RELATED TASKS

Using the EJB Development Environment: overview
Generating EJB deployed classes

Generating EJB deployed classes

When you generate deployed code, the enterprise beans are analyzed to ensure that the rules specified in the Sun Microsystems EJB specifications are met, as well as the rules specific to the EJB server. If any errors are found, error messages are issued to the Console log. Depending on the severity of the error, code generation might not begin.

Notes on generating deployed code:

- If you have an EJB group map defined for the enterprise beans, ensure that you only have one EJB group map loaded in the Map browser that is associated for the enterprise beans.
- If you want to generate deployed code for a CMP bean, you must ensure that the CMP bean is mapped before generating the code. Information about mapping the CMP bean is found in the topic “Generating a schema and mapping from an EJB group”.

Notes on regenerating deployed code:

- You do *not* need to regenerate the deployed code for an enterprise bean in the following situations:
  - An enterprise bean method is changed
  - A user-added method of the enterprise bean is changed.
- You *do* need to regenerate the deployed code for an enterprise bean in the following situations:
  - The home or remote interface (or base interface) is added, removed, or changed.
  - A home or remote method is added to (or removed from) the home or remote interface.
  - The method signature is changed for a method that exists in the home or remote interface.
  - A CMP field is added, deleted, or changed.
  - A key field designation is changed.
  - The mapping is changed.
  - An access bean is added or deleted.
  - A child enterprise bean is added or deleted that exists in an EJB inheritance relationship.
  - An association is added, changed, or deleted.
- Regenerating deployed code generation may omit stub generation. The first time you generate deployed code for an enterprise bean may take a while and you may see a progress dialog that indicates that the tool is generating stubs. However, the deployed code generation tool has been optimized for performance for subsequent deployed code generation when there is existing deployed code that is referenced by an enterprise bean. If you have made some changes to your bean-related classes and you need to regenerate deployed code,

the tool will only regenerate stubs and tie classes for those remote interfaces whose signature have changed since the last code generation. As a result, if you have not made changes to the signature of the enterprise bean's remote or home interfaces (or the interfaces that the remote or home interface extends), the stub regeneration phase of the deployed code generation is bypassed. This is normal behavior and is not a cause for concern.

However, in some rare situations where your enterprise bean interface references non-EJB remote interface type objects in the arguments or return types of the enterprise bean's remote interfaces, you may find that the stubs of those non-EJB remote interface types may not have been regenerated. When this occurs, you may run into a run-time error indicating that some stub classes are not found. If this happens, you should try deleting the deployed code and then regenerating it.

(Note that for the purpose of stub generation, an interface type is considered to be a remote interface type when the interface extends java.rmi.Remote, or all of its methods throw java.rmi.RemoteException or the super interfaces of java.rmi.RemoteException.)

To generate EJB deployed classes:

1. In the Enterprise Beans pane of the EJB page, select an EJB group or an enterprise bean.
2. Click mouse button 2 and select **Generate Deployed Code**.

   For each selected enterprise bean, the code-generation tool generates the home and EJBObject (remote) implementations and implementation classes for the home and remote interfaces, as well as the JDBC persister and finder classes for CMP beans. It also generates the Java ORB, stubs, and tie classes required for RMI access over IIOP, as well as stubs for the home and remote interfaces.

   If you selected an EJB group containing a CMP enterprise bean, or if you selected an individual CMP enterprise bean, the following items are also generated:

   - A create table string that is generated into the persister class.
   - A Persister implementation that maps to and from the table.

3. Once the classes have been generated, click on the  icon in the Types pane of the EJB page to see the generated types.

Note that if messages later appear in the status area of the EJB page that indicate errors exist in the deployed code of your enterprise bean, you can generally fix the errors by regenerating the deployed code for that bean.

Information about deleting EJB deployed classes is found in the topic "Deleting EJB groups and enterprise beans".

RELATED CONCEPTS

Overview of the EJB Development Environment

RELATED TASKS

Using the EJB Development Environment: overview
Generating a schema and mapping from an EJB group
Setting EJB deployment and control descriptor properties
Deleting EJB groups and enterprise beans

Chapter 7. Creating access beans and client applications

Creating or editing access beans

This topic assumes that you already understand access beans. For background information on how access beans are used in relation to enterprise beans, see the topic "Access beans: overview."

To create or edit an access bean:

1. In the Enterprise Beans pane, select the enterprise bean for which you want to create a new access bean or for which you want to edit an access bean.
2. From the **EJB** menu, select **Add** > **Access Bean**. The Create Access Bean SmartGuide opens.
3. In the **EJB group** and the **Enterprise bean** drop-down list boxes, ensure that the correct names appear.

   Note that in the **Access bean name** field, a default name appears. It is the same name that is specified in the **Enterprise bean name** field, except that the string `AccessBean` is appended to the name. For example, if you specify `MyBean` in the **Enterprise bean** field, the name `MyBeanAccessBean` is automatically specified in the **Access bean name** field. (You cannot modify the name in the **Access bean name** field.)
4. Do one of the following:

   - If the bean specified in the **Enterprise bean** field is a session bean, then `Java Bean Wrapper` is automatically selected as the access bean type in the **Access bean type** field. Click **Next**.
   - If the bean specified in the **Enterprise bean** field is an entity bean, in the **Access bean type** field, select the type of access bean that you want to create and then click **Next**.

   The Define Zero Argument Constructor page opens. You can use this page to specify a home interface method to correspond to the zero argument constructor of the access bean. Access beans are Java beans, so you can use them in visual programming tools such as the Visual Composition Editor. You can also convert the arguments in the home interface method to strings in the access bean constructor. This simplifies your visual development by allowing you to type in values for the arguments.
5. In the **Select home method for zero argument constructor** field, ensure that the correct home interface method is selected.

   - If you want to use the zero argument constructor to create a new instance, then select the create method.
   - If you want the zero argument constructor to find an existing instance, then select the findByPrimaryKey method.

   The home interface method that you select is used to instantiate the enterprise bean when the access bean is instantiated.
6. In the Initial Properties column, ensure that the check boxes are selected for those initial properties for which you want to select converters. (The Initial

© Copyright IBM Corp. 1998, 2000

Properties column lists the access bean properties, which are the arguments for the selected home interface method. You can select one or more properties and open a pop-up menu to help you manage them. The pop-up menu is described below at the bottom of this topic.)

7. For each initial property that is selected in the Initial Properties column, select one of the entries from the drop-down list in the Converter column.

   If you select **None**, no string converter is used for the selected initial properties.

   If you select **com.ibm.ivj.ejb.runtime.SimpleStringConverter**, the default string converter supplied with VisualAge for Java is used to perform string conversion for the selected initial properties. Note that the default string converter only supports the Java primitive types and their object wrappers.

   If you have created any string converters of your own, these string converters will appear as selectable items in the list. If you select one of your own converters, it must contain a type conversion for the selected type.

8. Do one of the following:

   - If you are creating a new Java bean wrapper access bean, click **Finish** to save your selections and generate the access bean.
   - If you are creating a new copy helper access bean or a rowset access bean, or if you are editing the properties of an existing copy helper or rowset access bean, click **Next**. The Select and Customize Bean Properties for Copy Helper page appears. (For an extended example of how to use copy helper and rowset access beans, see the topic "Example: Using copy helper and rowset access beans".)

9. In the Copy Helper column, select the checkbox beside each enterprise bean property for which you would like to have copy helper support. (Note that in the Enterprise Bean column, a pop-up menu is available to help you manage the properties. This pop-up menu is described below at the bottom of this topic.)

   To identify the enterprise bean properties that are selectable for the copy helper access bean, the SmartGuide performs introspection of the enterprise bean using the JavaBeans specification rules for detecting getter and setter accessors. If either the getter or setter method in the enterprise bean class throws an exception, the property will not be recognized as a valid Java bean property and the SmartGuide will not display it.

10. For each enterprise bean property that is selected in the Copy Helper column, select one of the entries from the drop-down list in the Converter column.

    If you select **None**, no string converter is used for the selected enterprise bean properties.

    If you select **com.ibm.ivj.ejb.runtime.SimpleStringConverter**, the default string converter supplied with VisualAge for Java is used to perform string conversion for the selected enterprise bean properties. Note that the default string converter only supports the Java primitive types and their object wrappers.

    If you have created any string converters of your own, these string converters

will appear as selectable items in the list. If you select one of your own converters, it must contain a type conversion for the selected type.

11. Click **Finish** to save your selections and generate the access bean.

Note that on both the Define Zero Constructor Argument page and the Select and Customize Bean Properties for Copy Helper page, if there are only a few initial properties or enterprise bean properties listed in the Initial Properties or Enterprise Bean columns, then it is generally easier to use the controls on the page to set any required properties. However, if there are numerous properties, the Initial Properties and Enterprise Bean columns provide a pop-up menu that can help you more easily manage a large number of properties. Information about the pop-up menu and its menu choices can be obtained pressing the F1 key on the Define Zero Constructor Argument page or the Select and Customize Bean Properties for Copy Helper page to open the contextual help information.

Information about deleting access beans is found in the topic "Deleting EJB groups and enterprise beans".

In addition to the information about access beans in this topic and in the topics listed below in the Related Concepts and Related Tasks, you can find more information about access beans in the paper entitled "Developing EJB Access Beans in VisualAge for Java", which is available from the VisualAge Developer Domain (VADD) Web site at the following URL:

www.ibm.com/software/vadd

RELATED CONCEPTS

Overview of the EJB Development Environment
Access beans: overview
Access beans: Java bean wrappers
Access beans: copy helpers
Access beans: rowsets

RELATED TASKS

Using the EJB Development Environment: overview
Creating client applications using access beans
Deleting EJB groups and enterprise beans

RELATED REFERENCES

Example: Using copy helper and rowset access beans

Organizing your client code

We recommend that you use a separate project to contain your client code. Instead of adding the server project to the classpath of the client project, we recommend that you include the client JAR file in the client classpath before executing the client application. This provides better code isolation between the client and server code.

A client JAR file contains all the classes and interfaces used to access enterprise beans. After the JAR file is created, you can use it to execute the client application.

RELATED CONCEPTS

RELATED TASKS

Using the EJB Development Environment: Overview
Exporting client-side code to client JAR files

Creating client applications using access beans

This topic contains two sections:

- "Creating client applications with access beans" shows you how to use access beans in developing your client applications.
- "Creating client applications without access beans" shows you how to develop client applications without using any access beans.

You will see how much easier it is to create client applications with access beans than to create them without access beans.

Creating client applications with access beans
When you use access beans to create your client application, you can use enterprise beans in much the same way that you would use standard Java beans.

To create a client application using an access bean:

1. Instantiate the access bean using one of its constructors. If you use the no-arg constructor, then you should also set the initial properties for the home interface method, which maps to the no-arg constructor.
2. Call methods on the instantiated access bean, as required. If you chose to create a copy helper or rowset access bean, then as required, you should also call the methods to refresh and commit the copy helper cache.

In the topic "Access beans: Java bean wrappers," an Employee example is used. The client code from the Employee example appears below and illustrates the two steps in creating a client application using an access bean:

```
EmployeeAccessBean aEmployee = new EmployeeAccessBean()
aEmployee.setInit_employeeNo ("100");
aEmployee.setName ("IBM");
aEmployee.setAddress ("1150 Eglinton Ave, Toronto");
```

If the client program sets copy helper attributes in an access bean and performs a get operation before calling commitCopyHelper(), then the access bean itself will return the data in the cache rather than the data in the enterprise bean. Generally, if the attribute resides in the cache, the access bean will get it from the cache. However, if you perform a get operation on a copy helper field that has not been set or retrieved, the cache will be refreshed from the enterprise bean.

Note that if you are going to call a finder that returns an enumeration, and you invoke it from a client program outside of a transactional scope, only the first five results will be returned in the enumeration. To return all of the results, you need to make sure that the finder method is invoked within a transaction. To do this, you can either call the finder method from a session bean method, where the method has container-managed transaction attribute, or create a user transaction in your client. For example:

```
// Get the initial context
java.util.Properties p = new java.util.Properties();
p.put(Context.PROVIDER_URL, "IIOP:///");
p.put(Context.INITIAL_CONTEXT_FACTORY, "com.ibm.ejs.ns.jndi.CNInitialContextFactory");
InitialContext initContext = new InitialContext(p);
```

```
// Look up a transaction
userTran = (UserTransaction)initContext.lookup("jta/usertransaction");
userTran.begin();
// Call the finder
// Assume that employeeHome has already been found, and has a method defined findAll()
Enumeration enum = employeeHome.findAll();
while (enum.hasMoreElements()) {
         // Process the enumeration
}
userTran.commit();
```

Additional information about custom finders is found in the topic "Implementing custom finders in home interfaces for CMP entity beans." Information about user transactions is found in the online documentation of the WebSphere Application Server.

Creating client applications without access beans
Prior to the availability of access beans, you needed to create a client application that employed enterprise beans using the following scenario.

A client application that used enterprise beans needed to first identify the enterprise bean instance it intended to use. The client performed this operation by using one of the create or find methods defined in the enterprise bean's home interface. These methods returned an object that implemented the enterprise bean's remote interface and could be used by the client application to access the enterprise bean. The tasks of obtaining an enterprise bean's home interface object and an enterprise bean's remote interface object are described below.

First, the client used the JNDI name services to locate a name server. The following sample code segment illustrates this step:

```
// Get the initial context
java.util.Hashtable properties = new java.util.Hashtable(2);
properties.put(javax.naming.Context.PROVIDER_URL, "iiop:///"); // local name server
properties.put(javax.naming.Context.INITIAL_CONTEXT_FACTORY,
   "com.ibm.ejs.ns.jndi.CNInitialContextFactory"); // IBM name services
javax.naming.InitialContext initContext = new javax.naming.InitialContext(properties);
```

This code segment used a provider URL of 'iiop:///'. This indicates that the name server was on the local host and listening on the default port. To specify a name server running on another host, a provider URL such as iiop://*hostname*/ was used, where *hostname* was the host name. To specify the port number that the name server was listening on, a provider URL such as iiop://*hostname:port* was used, where *port* was the port number.

Next, the client asked the name server to look up an entry for the deployed enterprise bean. In the following sample code segment, the initial context created above was used to look up the home instance. When this enterprise bean was deployed to the EJB server, it was configured to use a JNDI name of "HelloWorldBean", so that the name was looked up.

```
// Perform the lookup
HelloWorldHome helloHome = null;
try {
     Object obj = initContext.lookup("HelloWorldBean"); // JNDI name
     helloHome = (HelloWorldHome) javax.rmi.PortableRemoteObject.narrow(obj,
                                                HellowWorldHome.class);
}
```

```
catch (javax.naming.NamingException e) {
    System.out.println("Error retrieving the home interface: " + e);
    System.exit(0);
} // endtry
```

Finally, the client used the enterprise bean's home interface to obtain a reference to an enterprise bean instance. In the following code segment, the HelloWorldHome object (created above) was used to create a new HelloWorld session bean instance.

```
// Create a new HelloWorld
HelloWorld hello = null;
try {
    hello = helloHome.create();
}
catch (Exception e) {
    System.out.println("Exception creating new HelloWorld: " + e);
    System.exit(0);
} // endtry
```

The client application was then able to use the enterprise bean object.

RELATED CONCEPTS

Overview of the EJB Development Environment
Access beans: overview
Access beans: Java bean wrappers
Access beans: copy helpers
Access beans: rowsets

RELATED TASKS

Using the EJB Development Environment: overview
Implementing custom finders in home interfaces for CMP entity beans
Creating or editing access beans

RELATED REFERENCES

Example: Using copy helper and rowset access beans

Chapter 8. Testing enterprise beans

Creating an EJB server configuration

Once you have generated the deployed classes for your enterprise beans, you can create an EJB server configuration that consists of one or more EJB groups containing enterprise beans that you want to test run on the EJB server.

To create an EJB server configuration:

1. In the Enterprise Beans pane of the EJB page, select an EJB group.
2. Click mouse button 2 and select **Add to > Server Configuration** from the pop-up menu. Your EJB group is added to a server configuration and the EJB Server Configuration browser appears.

   In the EJB Server Configuration browser, an EJB server entry is added to the Servers pane. In the Enterprise Beans pane, your EJB group appears.

Once you have added your EJB group and enterprise beans to a server configuration, you may need to set some properties. Information about setting properties is found in the topic “Setting EJB server properties”.

RELATED CONCEPTS

Overview of the EJB Development Environment

RELATED TASKS

Using the EJB Development Environment: overview
Setting EJB server properties

Setting EJB server properties

If you just want to test and debug an enterprise bean and associated client application that you are running within VisualAge for Java, you should not need to change the default properties that are automatically established for the EJB server. However, if you are testing an entity bean, you must ensure that the database URL references a database that is configured on your system. Note that once set, only the database URL property for the EJB server is saved when you exit VisualAge for Java.

To set the properties for the EJB server:

1. In the EJB Server Configuration browser, in the Servers pane, select the EJB server.
2. Click mouse button 2 and select **Properties** from the pop-up menu. The Properties for EJB Server dialog appears.
3. Do one of the following:

   - If you have added an EJB group to the server configuration that contains at least one entity bean, then in the **Database Source** field, specify the database URL where the enterprise beans will be stored. Ensure that the URL of a database is specified that is configured on your system. For example:

© Copyright IBM Corp. 1998, 2000

```
jdbc:db2:MyDatabase
```

In the **Connection Type** field, select the database driver that you want to use.

- If you have added an EJB group to the server configuration that contains only one or more session beans, then in the **Database Source** field, type `InstantDB`.

In the **Connection Type** field, type `jdbc.idbDriver`.

4. If your database is set up so that a user ID and password is required to access the database, then specify a user ID and password in the **Database User ID** and **Database Password** fields.
5. In the **Trace Level** list box, specify whether you want a low-level, medium-level, or high-level trace displayed in the Console.
6. In the **Transaction Timeout** field, specify the number of seconds a transaction is allowed to proceed before it is automatically terminated. The default value is 120 seconds.
7. In the **Transaction Inactivity Timeout** field, specify the number of milliseconds a transaction can remain inactive before it is automatically terminated. The default value is 60,000 milliseconds.
8. If you want the container to get an exclusive lock on the enterprise bean when the "find by primary key" method is used, select the **Find for Update** check box.. The setting will take effect the next time the application server is started that is hosting the enterprise bean.

   This setting is useful for avoiding deadlock in the database. Deadlock can occur when two transactions run find methods, and then update methods, on the same enterprise bean. In this situation, the find method grants a shared lock on the enterprise bean, but the update method attempts to get an exclusive lock on the bean.
9. Click **OK**.

RELATED CONCEPTS

Overview of the EJB development environment

RELATED TASKS

Using the EJB development environment: overview
Starting EJB servers

Starting the database servers

If you want to run and test entity enterprise beans in the EJB Development Environment, you need to first start your database servers.

To start DB2 servers:

1. On the Windows® desktop, select **Start - Programs - DB2 for Windows - Command Window** to open a DB2 command window.
2. Issue the following two commands to ensure that the DB2 database server and the DB2 administration server are running:

```
db2start
db2admin start
```

If the two servers are not running, the commands start the servers. If the servers are already running, you are informed of this by a message.

RELATED CONCEPTS

Overview of the EJB Development Environment

RELATED TASKS

Using the EJB Development Environment: overview

Creating database tables

When you generate deployed code for a CMP entity enterprise bean, a set of create-table strings are generated into the persister class. This enables you to use the EJB Server Configuration browser to create database tables for whatever supported database you specified in the **Database URL** field of the Properties for EJB Server dialog.

To create a database table:

1. Launch the EJB Server Configuration browser. From the **EJB** menu, select **Open To > Server Configuration**. The EJB Server Configuration browser appears.
2. In the Servers pane, select the EJB server.
3. Click mouse button 2, and select **Create Database Table** from the pop-up menu.

   All of the database tables for the CMP entity enterprise bean are created (if they do not already exist). You can now test your enterprise bean and ensure that it works as expected.

Before changing the definition of a database table (by adding or deleting a column), you must first drop the table using a DB2 command before you can create a new table. To delete the existing table and create the new table in one step, use the Schema Browser to export the schema.

RELATED CONCEPTS

Overview of the EJB Development Environment

RELATED TASKS

Exporting schemas
Using the EJB Development Environment: overview
Generating a schema and mapping from an EJB group

Starting EJB servers

Once you have created an EJB server configuration and set any required properties, you need to start the EJB server before you can test your enterprise bean. (Information about creating an EJB server configuration is found in the topic “Creating an EJB server configuration.”)

Note that the number of EJB servers that you can start differs depending on the number of database connections configured on your database. An EJB server is often started with a pool of about 25 database connections. If you see an error message in the console log that indicates that the maximum number of database

connections is exceeded, you will need to reconfigure your database. The topic "Handling problems caused by DB2 connection or application limits" provides information about this situation.

To start the EJB server:

1. From the **Workspace** menu, select **Tools > WebSphere Test Environment**. The WebSphere Test Environment Control Center appears.
2. In the left pane of the Control Centre, click **Persistent Name Server**, then in the right pane, click **Start Name Server**. The Console window appears in the background and monitors the server. (Detailed information about the Persistent Name Server is found in the online help documentation for the WebSphere Test Environment component.)
3. In the status area of the WebSphere Test Environment Control Center, ensure that the message "Persistent name server is started" appears.
4. In the Servers pane of the EJB Server Configuration browser, select the EJB server and click the following icon to start the server:

   The Console window now also monitors the EJB server. When an EJB server is started, only enterprise beans that contain no errors, and for which deployed code has been generated, will actually run. In the Console, you can determine which enterprise beans are actually running.
5. In the Console, check that the EJB server is ready to process a call from the client by selecting the EJB server entry at the bottom of the All Programs pane and then looking for the following message in the Standard Out pane:

   ```
   Server open for business...
   ```

   (The message should appear in the last line at the bottom of the output text. However, it may take a moment or two to appear.)
6. You can change the source code of any enterprise bean in the EJB server configuration and debug it without stopping the EJB server. However, if you change the home or remote interfaces, you must do the following:

   a. Stop the EJB server. (Information about stopping the EJB server is found in the topic "Stopping EJB servers".)
   b. Regenerate the deployed code.
   c. Start the EJB server again.

   Notes: If you have problems starting the EJB server, in the Console window, select the EJB server process and examine the first error message that appears in the Standard Out pane. If the error message indicates that the maximum number of database connections or applications has been exceeded, refer to the topic "Handling problems caused by DB2 connection or application limits" for information on how to correct this problem. If, however, an error message indicates that the JDBC driver cannot be loaded, or if the debugger has opened, you should try creating a new project and importing the JDBC driver into the new project. In some situations, importing the JDBC driver may be necessary when working with entity enterprise beans.

Restarting Servers
In some situations, you may want to perform a restart on a server that is already running. Essentially, this resets the running server to the state it was in when it was first started.

To restart a server:

1. In the EJB page, select **EJB** > **Open To** > **Server Configuration**. The EJB Server Configuration browser appears.
2. In the Servers pane, select the server that you want to restart.
3. Click mouse button 2 and select **Restart Server** from the pop-up menu. In the Servers pane, an icon appears beside the server to indicate that it is running.

RELATED CONCEPTS

Overview of the EJB Development Environment

RELATED TASKS

Using the EJB Development Environment: overview
Creating an EJB server configuration
Stopping EJB servers
Handling EJB serialization problems
Handling problems caused by DB2 connection or application limits

Running the EJB test client

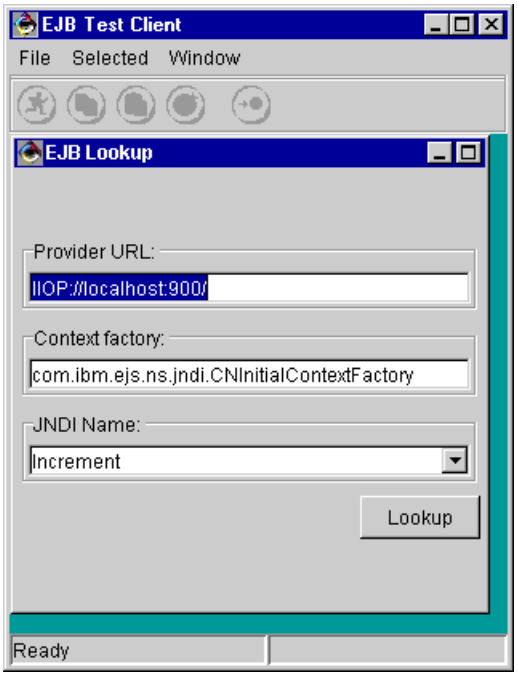
The EJB Development Environment provides a test client that you can use to test the home and remote interface methods of your enterprise beans. Using the test client, you can call the methods and pass user-defined arguments to test the methods and ensure that they work correctly.

This topic describes how to run the test client using its most basic features. For information about using the test client's more advanced features, see the topic "Using the advanced features of the EJB test client".

All of the Java projects found in the workspace are automatically added to the test client class path if you start the test client from either the EJB Server Configuration browser or the EJB page. When you run the test client, you can set breakpoints in your enterprise bean implementation and use the debugger as you would elsewhere in the VisualAge for Java IDE.

To run the test client:

1. Ensure that you have created an EJB server configuration that contains the enterprise beans that you want to test, and that you have started the EJB server. Information about creating EJB server configurations and starting EJB servers is found in the list of related tasks at the end of this topic.
2. In the Enterprise Beans pane of the EJB Server Configuration browser, select one or more enterprise beans that you want to test with the test client, then right-click and select **Run Test Client**. The EJB Test Client window opens.



3. In the **JNDI Name** list of the EJB Lookup frame, select the JNDI name of the enterprise bean that you want to test.

   Note that you do not need to open a separate test client window for each enterprise bean that you want to test. If you want to test an enterprise bean that you did not select in the Enterprise Beans pane of the EJB Server Configuration browser prior to starting the test client, simply type in the JNDI name of the bean.

4. Click **Lookup**. This directs the test client to construct an initial naming context and query the name server for the JNDI name selected in the JNDI list box. When the connection is established to the name server and an instance of the home interface is retrieved from the name server, the Home interface page for the enterprise bean appears.

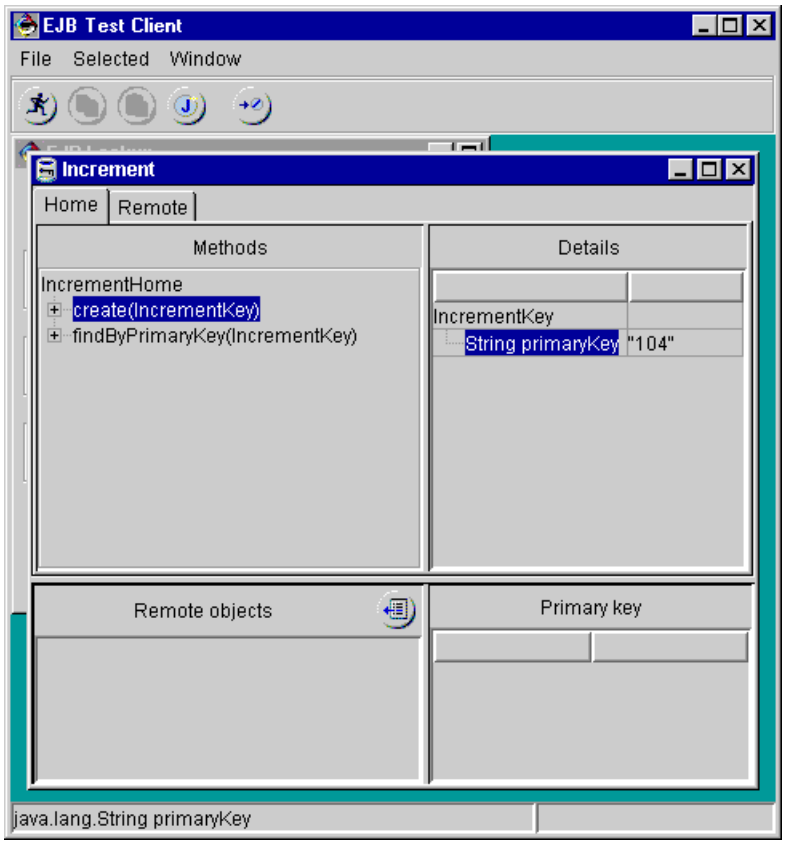
   In the figure below, a simple CMP entity bean named Increment is being tested. It is used to increment a counter. Its home interface contains a create method and a findByPrimaryKey method. Both of these methods require an IncrementKey argument. The Increment bean also contains a findGreaterThan method, which requires an int argument.

5. In the Methods pane, select the method that you want to test. In the Details pane, a tree table is presented of the method argument objects.

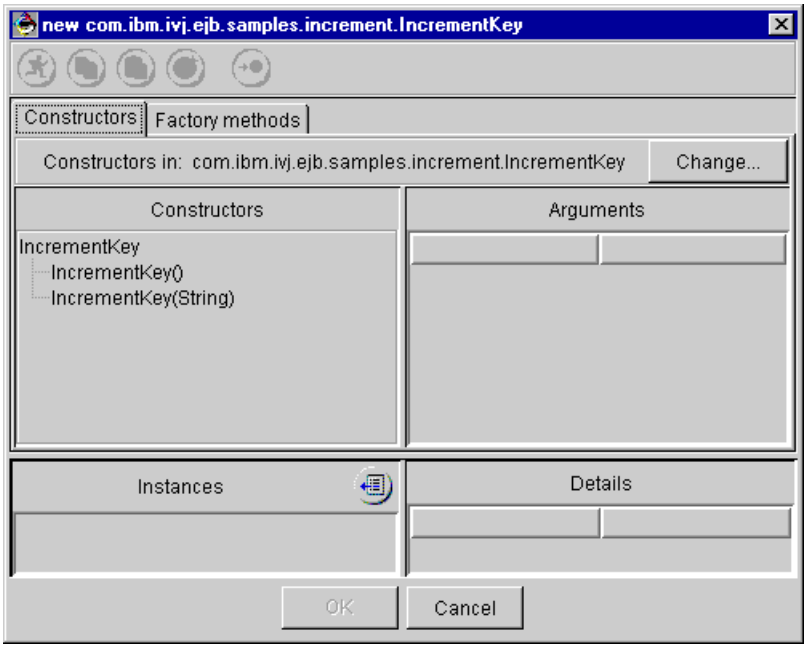
   If the method only takes a single argument, the argument appears as the root of the tree table. If the method takes more than one argument, the root of the tree table defaults to `Arguments` and each method argument appears as a child node of the root. For an argument that has a primitive type, such as int, float, or boolean, a simple text field or choice property editor is displayed. For an argument that has a complex type, the object is displayed as a subtree where the object's fields, properties, methods, and superclass comprise the child nodes. In the figure below, the key object is the root of the tree table because it

is the only argument required for the method. The child node displayed under the key object is a field named `primaryKey` and its type is java.lang.String. (By default, only member fields are displayed for EJB key objects.)

6. If you want to edit the value of a field, in the right column of the Details pane beside the field, position your cursor in the text area and type in the new value for the field:



7. The test client attempts to construct method arguments for you by using the default constructor for the method parameter type. In this case, the argument was created automatically because IncrementKey has a default constructor. If the parameter type does not have a default constructor and you want to create an object to use as the method argument, then on the tool bar, click **Construct new object** to create the argument object. The Constructors page of the object constructor dialog appears.



The Constructors page shows the constructors defined for the complex type. If you want to edit a constructor argument, do the following:

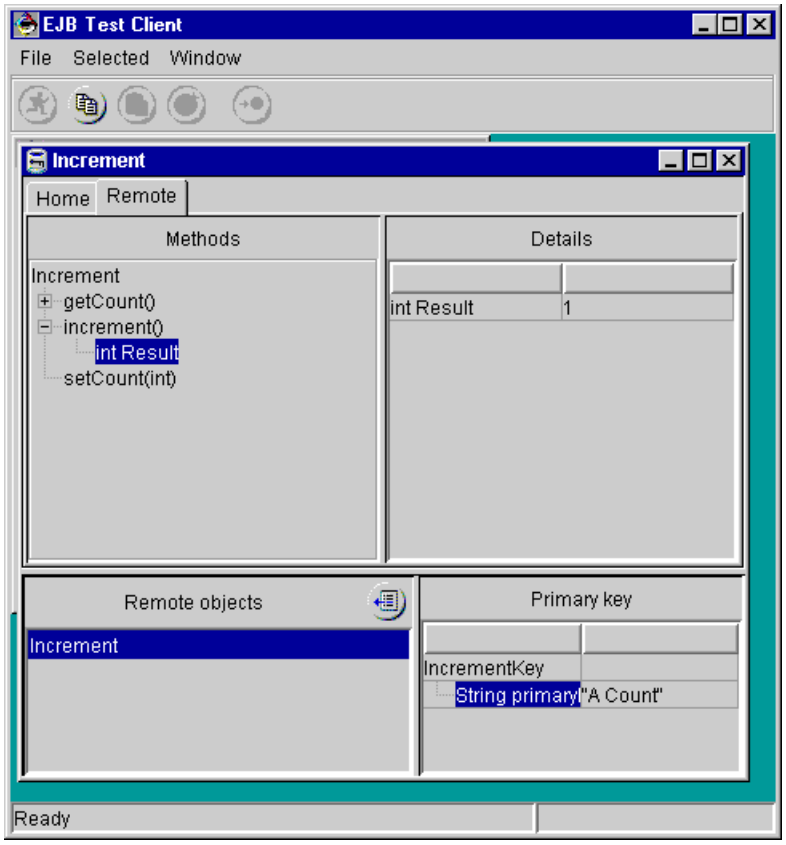
a. In the Constructors pane, ensure the constructor is selected.
b. In the Details pane to the right of the argument, type in a value for the argument in the text area.
c. Click the **Invoke** button to call the constructor and create the new instance. The new object is displayed in the Instances pane at the bottom of the Constructors page. (For example, in the above figure, you could specify `A Count` as the value for the String argument, then create a new instance.)
d. If you want to modify the new object, select it in the Instances pane, and then in the right pane, edit the object.
e. Click **OK** to save the changes and close the Constructors page.

8. On the Home interface page, ensure the correct method is selected, then click the **Invoke** button.

   This sends the method call to the EJB server. In our example, the create(IncrementKey) method call causes the EJB server to attempt to construct a new Increment enterprise bean with the IncrementKey we provided. The resulting remote interface object is used by the test client for testing remote methods. (Alternatively, you could have run the findByPrimaryKey method to locate an existing enterprise bean.)

   When the remote interface object is retrieved, the Remote interface page appears. In the Remote Interface page, the Methods pane contains the list of methods defined by the remote interface.

9. In the Methods pane, select the method that you want to test, then enter the required parameters using the same approach you used on the Home interface page.
10. Click the **Invoke** button to send the method to the enterprise bean. The result of the method call is displayed as a child node of the method in the Methods pane. When the method result is selected, the object is displayed in the Details pane. This allows you to view the attributes of the results object, such as fields and properties.



RELATED CONCEPTS

Overview of the EJB Development Environment

RELATED TASKS

Using the EJB Development Environment: overview
Creating an EJB server configuration
Starting EJB servers
Using the advanced features of the EJB test client

Using the advanced features of the EJB test client

The basic features of the EJB test client are described in the topic "Running the EJB test client". The advanced features of the test client are described below.

Connecting to remote servers

The test client can be used to connect to a remote name server. This allows you to

access enterprise beans that are running on an EJB server on another machine, such as the WebSphere Application Server.

To connect to a remote server, do the following:

1. In the Enterprise Beans pane of the EJB page, select any enterprise bean. (This is required to open the test client. Since you are going to test enterprise beans that reside on a remote machine, it doesn't matter which enterprise bean you select locally.)
2. From the **EJB** menu, select **Run Test Client**. The EJB Test Client opens and displays the EJB Lookup frame.
3. In the **Provider URL** field, specify the host name (and optionally, the port number) for the remote name service. The provider URL must have the following form (where *hostname* is the remote name server's host name and *port* is the port number on which the remote name server is listening):

   iiop://*hostname:port*/
4. In the **JNDI name** field, type the name of the enterprise bean you want to test that is running on the remote machine.

Working with multiple remote interface objects
Using the test client, you can create and work with several different remote interface objects at the same time. To facilitate this, on both the Home interface and Remote interface pages of the test client, you'll find a Remote objects pane. On the Home interface page, you can create (or find) many different remote interface objects that are added to the Remote objects pane on the Remote interface page. If the enterprise bean is an entity bean, the bottom half of the page consists of two panes. The left Remote objects pane contains the remote objects list, and the right Primary key pane contains a tree table of the enterprise bean's primary key object. To change the target of the remote method calls, select a different remote interface object from the list. Subsequent remote method calls will be directed to that object. To remove a remote object from the list, click the **Drop selected** button in the Remote objects pane. Note that this only removes the entry from the list. It is not the same as EJBObject.remove(). Also note that when you invoke EJBObject.remove(), the entry is not automatically removed from the remote objects list.

Viewing as declared or actual type
If you want to show the declared type of an element, select the element and then right-click and choose **View as declared type**. If you want to show the actual type of an element, select the element and then right-click and choose **View as actual type**.

Viewing additional object attributes
Sometimes it is useful to view object attributes (fields, properties, etc.) that are not shown by default. For example, when the home interface for an entity bean is displayed, by default only the methods declared in the home interface are shown. If you want to invoke a method in the super interface, EJBHome, you would need to change the view so that the super interface is shown. To do this, right-click on the root of the home interface tree in the Details pane, then select **Show object features** from the popup menu. The Show Object Features dialog is displayed, which contains checkboxes for member fields, static fields, normal properties, expert properties, etc. To view the super interface, select the **Extends** checkbox and click **OK**. The Home interface page now contains two children of the root: 'Methods' and 'extends EJBHome'. When you expand the 'extends EJBHome'

node, it will also contain two children: 'Methods' and 'extends Remote'. When you expand the Methods node of EJBHome, you see methods declared in that interface.

Note that you can use the Show Object Features dialog to display properties of an enterprise bean. This is not used as the default view of an enterprise bean because each time the view is displayed, the get method for the property is invoked. Since the get method is a remote method call, this view would operate too slowly.

Copying and pasting Java objects
Using the copy, paste, and clipboard tools of the test client, you can reuse the same Java object in two different places. For example, the method EJBHome.remove(Handle) requires a javax.ejb.Handle object. You can obtain an instance of this object by calling the method EJBRemote.getHandle(). (Information on how to view the super interface is found in the previous section.) Select the getHandle() method in the Remote interface page, then click the **Invoke** button. The Result object is an instance of javax.ejb.Handle that refers to your enterprise bean object. Select the object instance, then click the **Copy** button to copy this object instance to the clipboard. Note that the copy operation copies a reference to the object instance. It is not a clone operation. Return to the Home interface page and select the remove(Handle) method in the EJBHome interface display. Select the root of the argument tree, 'Handle', which is initially a null value, then click the **Paste** button to set the Handle argument to the Handle object reference stored in the clipboard. (Note, however, that the test client does not use the Windows clipboard, so you can only copy and paste within a single instance of the test client, not between two instances of the test client.)

It is important to remember that the **Copy** and **Paste** buttons operate on an object reference, not on an object value. To demonstrate the difference, run the test client against the Increment enterprise bean used in the Increment sample. On the Home interface page, select the create(IncrementKey) method. In the Details pane, edit the primaryKey field to contain the value `A Count`. Copy the IncrementKey argument object (the root object in the Details pane), then select the findByPrimaryKey(IncrementKey) and paste the IncrementKey object into the Details pane. Now edit the primaryKey field to contain the value `Another Count`. Select the create(IncrementKey) method again. Note that the IncrementKey argument object has changed. The primaryKey field reads `Another Count`. This is because both methods are using the same Java object as their argument. An edit in one place results in a change to the object view in both places.

Since the test client provides the ability to copy and paste references to Java objects, you can easily test enterprise beans with associations. For example, you may have a Department enterprise bean and an Employee enterprise bean, with a one-many association between them. In the test client, you can first look up the Department home and then use an ejbCreate() method to create a Department. In the same test client, you can then look up the Employee home, and use its ejbCreate() method to create an Employee. In the Employee window, in the Remote objects pane, right-click on the Employee object and select **Copy**. In the Department window, in the left pane, select the method addEmployee(Employee). In the right pane, right-click on the argument, and select **Paste**. This will paste the reference to the Employee object into the method. When you click the **Invoke** button, the method will be called with the Employee as the argument.

Replacing objects with null references
The test client provides an **Assign null** button that you can use to replace an object with a null reference. This is sometimes useful for testing how well your implementation deals with null method arguments.

Constructing new objects using the Factory methods page
In the test client, a Factory methods page provides you with another way of constructing new objects. Some class definitions, such as a Java interface or a class like java.lang.Class, have no public constructors. Sometimes these objects must be constructed by calling a static method such as forName() in java.lang.Class. The title of the Factory methods page shows you the name of the class representing the type of object to be constructed (the target class). Initially, the Factory methods page displays a list of static methods declared in the target class whose return type can be assigned to the target class. In the case of java.lang.Class, the Factory methods page shows the methods forName(String) and forName(String, boolean, ClassLoader). Each of these static methods returns an instance of java.lang.Class. If the factory method you want to use is declared in another class, click the **Change** button to open the Change class dialog, specify a new class name in the **Use factory methods in** field, then click **OK**. The Factory methods page will be loaded with a list of static methods from the new class whose return type is assignable to the target type.

RELATED CONCEPTS

Overview of the EJB Development Environment

RELATED TASKS

Using the EJB Development Environment: overview
Creating a server configuration
Starting EJB servers
Running the EJB test client
Handling EJB serialization problems

Running your own client application

In the EJB Development Environment, you can use the EJB test client to run and test enterprise beans from within either the EJB Server Configuration browser or the EJB page. However, if you have developed your own EJB client application and you want to run it to access and test an enterprise bean, you must first follow the normal setup procedure for a Java program in the IDE by setting up your class path.

Information about developing your own client application is found in the topic "Creating client applications using access beans".

To start your own client application:

1. Ensure that the Persistent Name Server and the EJB server are running. (The Persistent Name Server can be started from the WebSphere Test Environment Control Centre. Information about starting the Persistent Name Server is found in the online documentation for the WebSphere Test Environment feature.)
2. In the Enterprise Beans pane of the EJB Server Configuration browser, select the enterprise bean which you will access with your own client application. (If the enterprise bean does not appear in the Enterprise Beans pane of the EJB Server Configuration browser, you must add the enterprise bean to an EJB server configuration by following the instructions in the topic "Creating an EJB server configuration".)
3. Add the IBM WebSphere Test Environment project to the class path by doing the following:

a. In the Workbench Projects page or other page where your client resides, select your client class and click mouse button 2, then select **Properties**. The Properties dialog appears.
b. Click the **Class Path** tab.
c. Beside the **Project Path** field, click the **Compute Now** button.
d. Since the **Compute Now** button does not automatically update the class path with the required WebSphere Test Environment project, beside the **Project Path** field, click the **Edit** button. The Class Path dialog appears.
e. Select the **IBM WebSphere Test Environment** project, then ensure that the **IBM EJB Tools** project and the **IBM Enterprise Extension Libraries** project are selected and click **OK**.
f. At the bottom of the Properties dialog, select the **Save in repository (as default)** check box and click **OK**.

4. Select your client class, then click mouse button 2 and select **Run - Run main**.
5. In the All Programs pane of the Console, select the client process. In the Standard Out pane of the Console, the client output appears.

Note that once your client is running, you can set breakpoints in your client and server and use the local debugger as you normally would.

RELATED CONCEPTS

Overview of the EJB Development Environment

RELATED TASKS

Using the EJB Development Environment: overview
Creating client applications using access beans
Creating an EJB server configuration
Starting EJB servers

Stopping EJB servers

This topic discusses how to stop or remove EJB servers.

Stopping an EJB server

To stop an EJB server:

1. In the EJB Server Configuration browser, in the Servers pane, select the EJB server that you want to stop.
2. Click on the  icon to stop the server. The Console page indicates that the server has been terminated.

Removing EJB servers

To remove an EJB server:

1. In the EJB Server Configuration browser, in the Servers pane, select the EJB server that you want to remove.
2. Click on the  icon to stop the server (if it is not already stopped).
3. Select the EJB server again, click mouse button 2, and select **Remove Server** from the menu. The EJB server is removed from the list of servers in the Servers pane.

RELATED CONCEPTS

Overview of the EJB Development Environment

RELATED TASKS

Using the EJB Development Environment: overview
Creating an EJB server configuration

Chapter 9. Managing EJB groups and enterprise beans

Validating enterprise beans

The EJB Development Environment automatically and seamlessly verifies that your enterprise bean code is consistent and that it conforms to the rules defined by the Sun EJB specification. Code verification occurs whenever an enterprise bean or its properties are changed. If any errors are detected, an error or warning symbol appears beside the enterprise bean and a message appears in the status bar at the bottom of the EJB page.

The EJB Development Environment also automatically verifies that access beans are constructed correctly and that they are consistent with their associated enterprise beans. Code verification occurs whenever you create or edit access beans.

Although error and warning messages are generally only displayed in the status bar, there may be occasions when you want to initiate a validity check on your enterprise beans and direct the messages to the Log, such as when a message is too long to view in its entirety in the status bar.

To validate your enterprise beans and direct the messages to the Log:

1. In the Enterprise Beans pane, select the EJB groups or enterprise beans for which you want to validate code.
2. From the **EJB** menu, select **Verify**. If there are any problems with the enterprise bean code, the Log window is opened and displays the text of the error or warning message.

Note that if messages later appear in the status area of the EJB page that indicate errors exist in the deployed code of your enterprise bean, you can generally fix the errors by regenerating the deployed code for that bean. Information about generating deployed code is found in the topic “Generating EJB deployed classes.”

RELATED CONCEPTS

Overview of the EJB Development Environment

RELATED TASKS

Generating EJB deployed classes

Moving or copying enterprise beans between repositories

To move or copy enterprise beans from one repository to another, do the following:

1. Save schemas and maps.
2. In the EJB page of the Workbench, version the EJB groups containing the enterprise beans you want to move or copy.
3. Go to the Projects page in the Workbench, then version the associated project.
4. Select the project that contains the packages associated with your EJB group or enterprise bean. These packages are the:
   - Reserved package
   - Package containing the enterprise bean classes and interfaces
   - Package containing the enterprise bean schemas and maps

© Copyright IBM Corp. 1998, 2000

Note that if these packages are spread across more than one project, you can expand these projects and do a multiple selection of the packages. However, we recommend that these associated packages be kept in a single project.

5. Version the project and export it as a .dat file into the target repository. (To open the export dialog, select **File > Export** in the Packages page.)

   Note: If the target repository is on a different machine that is not running EMSERV, or the target repository on an EMSERV machine is not hooked up to EMSERV, once you export the project to the .dat file, move the .dat file to the other machine. Then, import the .dat file into the repository where you want to locate the project. (To open the import dialog, select **File > Import** in the Packages page.)

RELATED CONCEPTS

Editions and versioning

RELATED TASKS

Exporting to another repository
Importing from another repository

Versioning EJB groups and enterprise beans

Generally, it is a good idea to version, release, and share EJB groups and enterprise beans at the project level by using the steps outlined in "Moving or copying enterprise beans between repositories." However, you can create different versions of your EJB groups and enterprise beans, which are automatically released so that they can be viewed and accessed by other members of your development team.

When you version an enterprise bean, any corresponding Java type (class or interface) that was generated from the enterprise bean is also versioned, unless the Java type was versioned separately prior to versioning the enterprise bean.

Note that the topic "Editions and versioning" recommends that you keep all the parts of the enterprise beans you are developing in one project. This makes it easier to keep all the parts synchronized and then to version them. Also, helpful information is found in the topic "EJB development in a team environment".

To version existing EJB groups and enterprise beans:

1. In the Enterprise Beans pane of the EJB page, select one or more EJB groups or enterprise beans you want to version.
2. Click the right mouse button to launch the pop-up menu and select **Manage > Version**. The Versioning Selected Items dialog appears.
3. Do one of the following:
   - If you want to automatically assign a single version name to the selected EJB groups or enterprise beans that is unique and logically incremental, ensure that the **Automatic** radio button is selected. This is the recommended action.
   - If you want to manually assign a single new version name of your own to the selected EJB groups or enterprise beans, select the **One Name** radio button and type a name for the new version in the entry field. This new version name will also be assigned to all enterprise beans and generated Java types that reside under any selected groups.
   - If you want to manually assign individual new version names of your own to each of the selected EJB groups and enterprise beans, then select the **Name**

Each radio button. The Versioning dialog appears, and displays the name of the first EJB group, enterprise bean, or generated class that appears in the block of EJB groups and enterprise beans selected in the Enterprise Beans pane. In the entry field, accept the default or type in a new name, then click **OK**. The Versioning dialog reappears and displays the next EJB group or enterprise bean in the block of selected EJB groups and enterprise beans. This process will continue until all enterprise beans and generated Java types that reside under selected groups have been individually named.

4. Click **OK**.

Whenever you version an EJB group, you should first save and version the schema and map classes, where applicable, to the reserved package. The next time the EJB group is loaded from the repository, the version of the schema and map classes that will be loaded will automatically be in synchronization.

RELATED CONCEPTS

Overview of the EJB Development Environment
Editions and versioning
EJB development in a team environment

RELATED TASKS

Using the EJB Development Environment: overview
Creating open editions of EJB groups and enterprise beans
Moving or copying enterprise beans between repositories

Releasing EJB groups and enterprise beans

Generally, it is a good idea to version, release, and share EJB groups and enterprise beans at the project level by using the steps outlined in "Moving or copying enterprise beans between repositories." However, you can release enterprise beans into the EJB group to which they belong and EJB groups into the project to which they belong. Releasing an EJB group releases the corresponding reserved package into the project to which it belongs.

To release EJB groups or enterprise beans:

1. In the Enterprise Beans pane of the EJB page, select the EJB groups or enterprise beans you want to release.
2. Click the right mouse button and select **Manage > Release**.

When you release selected EJB groups or enterprise beans, the corresponding generated classes do *not* also get released.

RELATED CONCEPTS

Overview of the EJB Development Environment
Relationship between the EJB page and the reserved package

RELATED TASKS

Using the EJB Development Environment: overview
Moving or copying enterprise beans between repositories

Creating open editions of EJB groups and enterprise beans

When you create new EJB groups and enterprise beans, they are automatically created as open editions. However, you can also create open editions for any existing EJB groups and enterprise beans that are versioned.

To create open editions of versioned EJB groups or enterprise beans:

1. In the Enterprise Beans pane of the EJB page, select a versioned EJB group or versioned enterprise bean for which you want to create an open edition.
2. Click mouse button 2 and select **Manage > Create Open Edition**. In the Enterprise Beans pane, an open edition of the versioned EJB group or enterprise bean appears. If you selected a versioned enterprise bean, an open edition was automatically created for the EJB group to which it belongs. (If an enterprise bean is modified and the group is already versioned, a scratch edition is created for the group automatically.)

Note that the associated source code packages are not affected by creating an open edition of an EJB group or enterprise bean. If you plan to modify or regenerate code for your enterprise beans, you will need to create open editions of the source code packages.

RELATED CONCEPTS

Overview of the EJB Development Environment
Editions and versioning
EJB development in a team environment

RELATED TASKS

Using the EJB Development Environment: overview
Versioning EJB groups and enterprise beans

Replacing EJB groups and enterprise beans

In the EJB page, you can replace an EJB group or enterprise bean with a different edition of the same EJB group or enterprise bean. You can do this in one of two ways:

- Replace the EJB group or enterprise bean with the most recent previous edition of the same EJB group or enterprise bean.
- Replace the EJB group or enterprise bean with a specified edition of the same EJB group or enterprise bean.

When the replacing enterprise bean is a versioned edition, its associated generated Java types are also brought into the workspace to replace the Java types that are associated with the enterprise bean being replaced. Other classes on which your enterprise beans are dependent are not loaded, however.

Replacing EJB groups or enterprise beans with the most recent previous edition
To replace an EJB group or enterprise bean with the most recent previous edition of the same EJB group or enterprise bean:

1. In the Enterprise Beans pane, select the EJB group or enterprise bean you want to replace.
2. Click mouse button 2 and select **Replace With** > **Previous Edition**. A dialog box displays the progress of the replacement and the selected EJB group or enterprise bean is automatically replaced with the most recent previous edition.

Replacing EJB groups or enterprise beans with a specified edition
To replace an EJB group or enterprise bean with a specified edition of the same EJB group or enterprise bean:

1. In the Enterprise Beans pane, select the EJB group or enterprise bean that you want to replace.
2. Click the right mouse button and select **Replace With** > **Another Edition**. A dialog appears.
3. In the top pane of the dialog, choose the edition of the EJB group or enterprise bean you want to use to replace the selected EJB group or enterprise bean.
4. Click **OK**.

RELATED CONCEPTS

Overview of the EJB Development Environment
Editions and versioning

RELATED TASKS

Using the EJB Development Environment: overview
Creating open editions of EJB groups and enterprise beans

Deleting EJB groups and enterprise beans

Occasionally, you may find that you need to delete one or more EJB groups or enterprise beans. In some cases, you may want to only delete the access beans or deployed code that are associated with enterprise beans.

If you want to delete versioned enterprise beans, any associated Java types that were generated from the enterprise beans are only deleted if the enterprise beans contain the correct version signatures for the Java types. If the enterprise beans contain version signatures that do not match the corresponding version signatures of the Java types, the Java types are not deleted.

Note that the packages that are used to hold the generated Java types are not deleted with the associated enterprise bean. You must manually delete unused packages from your project when the generated classes in those packages are removed.

To delete EJB groups, enterprise beans, access beans, or deployed code:

1. In the Enterprise Beans pane of the EJB page, select one or more EJB groups or enterprise beans.

   If you are deleting an enterprise bean, the reserved package must be an open edition. The source package must be an open edition.
2. Click the right mouse button and select **Delete**. A dialog appears that gives you the following options:
   - **All** Delete the selected bean and all its associated generated types.
   - **Bean Only** Delete only the bean and not all its associated generated types.
   - **Access Bean** Delete only the access bean associated with the selected enterprise bean.
   - **Deployed Code** Delete only the generated deployed code for the selected enterprise bean.
   - **Cancel** Close the dialog without deleting anything.

RELATED CONCEPTS

Overview of the EJB Development Environment

RELATED TASKS

Using the EJB Development Environment: overview

Changing ownership of EJB groups or enterprise beans

In the EJB Development Environment, change control is based on ownership. At times, you might want to reassign ownership of EJB groups or enterprise beans; for example, when someone is leaving the development team. To make this change, you must be the EJB group owner.

To change the ownership for EJB groups or enterprise beans:

1. In the Enterprise Beans pane of the EJB page, select the EJB groups or enterprise beans for which you want to change ownership.
2. Click mouse button 2 and select **Manage > Change Owner**. A list of eligible users appears.

   (A user can be added to the list of eligible users for a group by selecting the group, clicking mouse button 2, and selecting **Manage >Add User to Group**.)
3. Select the name of the user who will be the new owner and click **OK**.

Note that you should also change the ownership of the associated package and project, as well as the schema and map storage classes. These changes are necessary if you need to modify the enterprise beans and generate new deployed code.

RELATED CONCEPTS

Overview of the EJB Development Environment
Ownership and team roles: overview

RELATED TASKS

Using the EJB Development Environment: overview
Changing a program element's owner
Adding users to EJB groups

Adding users to EJB groups

Before developers can own, create, or release enterprise bean classes, they must be members of the appropriate EJB group.

Only the EJB group owner can change the membership of the EJB group. To determine who the owner of an EJB group is, do one of the following:

- Select the EJB group in the **Managing** page of the Workbench window, and look for the owner marker (>) in the **Package Group Members** pane.
- Select the EJB group, click mouse button 2, and select **Properties**. The Info page shows you the name of the owner.

Users must be in the repository user list before they can be added to an EJB group. The topic "Adding users to the repository user list" provides more details.

To add users to an EJB group:

1. In the Enterprise Beans pane of the EJB page, select the EJB group or groups to which you want to add users.
2. Click mouse button 2 and select **Manage > Add User to Group**. The Add Users dialog shows a list of users in the repository list who are not yet members of the EJB group or groups.
3. Select one or more user names, and click **OK**.

The users are added to the selected EJB group or groups. To verify, select the reserved package for each EJB group in the **Managing** page of the Workbench window, and look at the list of names in the **Package Group Members** pane.

The new user is added only to the reserved package, not to the source packages. You will need to do this with the VisualAge for Java development tools. For some suggestions on how to develop enterprise beans in a team environment, see the "best practices" paper entitled "EJB Team Development in VisualAge for Java", which is available from the VisualAge Developer Domain (VADD) Web site at the following URL:

http://www.ibm.com/software/vadd

RELATED CONCEPTS

Overview of the EJB Development Environment
Ownership and team roles: overview

RELATED TASKS

Using the EJB Development Environment: overview
Adding members to a package group
Adding users to the repository user list

Keeping EJB groups in synchronization with schemas and maps

The meta data for schemas and maps are stored in separate classes from the EJB group and enterprise beans. When an EJB group is loaded, the corresponding schema and map are loaded automatically into the Map and Schema browsers if their storage classes are already loaded or loaded at the same time with the EJB group (that is, within the reserved package.)

In a team environment where enterprise beans are shared, it is recommended that you share enterprise beans at the project level. In this way, the EJB group and corresponding map and schema are more easily synchronized, even if they were not saved to the reserved package.

It is easier to keep the schema and map synchronized with the EJB group if they are saved to storage classes within the reserved package. If you choose not to save the schema and map to the reserved package, it is recommended that you save them to a package in the same project as the reserved package.

It is important that you save the schema and map before versioning the EJB group. Otherwise, any changes you made to the schema and map since the last time you saved them will be lost.

In a team environment where enterprise beans can be shared, the project owner is responsible for merging all of the enterprise beans and all of the corresponding schema classes and map classes, then creating an edition of the project so that the

team members can share the project. An individual team member cannot replace an edition of an enterprise bean without proper authority to the bean class and all corresponding classes.

RELATED CONCEPTS

Overview of the EJB Development Environment
Editions and versioning

RELATED TASKS

Using the EJB Development Environment: overview

Chapter 10. Exporting EJB groups and enterprise beans

Exporting enterprise beans to EJB or deployed JAR files

After you have tested your enterprise beans, you can export them to an EJB JAR file or a deployed JAR file. (You can also export client-side code to a client JAR file, which is described in the topic "Exporting client-side code to client JAR files.")

Choosing the appropriate exportation method
If you plan to deploy your enterprise beans to a non-IBM EJB server, export them to an EJB JAR file. This file contains enterprise bean code, such as the enterprise bean class, home and remote interfaces, deployment descriptor (.ser file), and manifest. It does not, however, contain deployed classes.

If you plan to deploy your enterprise beans to the WebSphere Application Server, *Enterprise Edition*, export them to an EJB JAR file composed specifically for the Component Broker (CB) deployment tools. (If your workstation does not have the WebSphere Application Server, Enterprise Edition installed, you can launch the CB deployment tools separately after exportation.) Depending on implementation, some data types may have to be remapped. For more information, see the topic "Default type mappings for deployment to Component Broker."

If you plan to deploy your enterprise beans to the WebSphere Application Server, *Advanced Edition*, export them to a deployed JAR file. This file contains all of the code defined in the EJB JAR file as well as deployed classes that are specific to the WebSphere Application Server, Advanced Edition. At deployment time, WebSphere will recognize that the deployed classes already exist; it will not try to regenerate them.

Note: If you are using a "where clause" finder helper string in your finder helper class, you must use VisualAge for Java to generate the deployed classes for the bean and you must export the bean to WebSphere Application Server using a deployed JAR file. You cannot use an EJB JAR file to export the bean because the WebSphere Application Server deploy tool cannot deploy beans that use the "where clause" finder.

Packaging considerations
If an enterprise bean contains a custom finder class, the custom finder class will be exported automatically.

Enterprise beans that exist in an inheritance or association relationship must be packaged in the same JAR file.

You can export entity beans and session beans in the same JAR file. If you are deploying a JAR file to the WebSphere Application Server, *Advanced Edition*, your session beans and entity beans can reside in the same container. However, if you are exporting to the WebSphere Application Server, *Enterprise Edition*, the session beans must reside in one container and the entity beans must reside in a different container.

Generally, any EJB server run-time files are automatically excluded from JAR files. However, if an access bean with one or more copy helpers exists for an enterprise bean that is to be exported to an EJB JAR file, the class

© Copyright IBM Corp. 1998, 2000

com.ibm.ivj.ejb.runtime.CopyHelper is also exported to the JAR file. If you later delete the enterprise bean from the workspace and then import it again in an EJB JAR file, you will be prompted about whether you want to create an open edition for com.ibm.ivj.ejb.runtime. Make sure to specify that you do *not* want to create an open edition.

Exporting the beans

To export enterprise beans to a deployed JAR file or to an EJB JAR file:

1. In the Enterprise Beans pane of the EJB page, select one or more EJB groups or enterprise beans.

   If you are exporting enterprise beans to an EJB JAR file intended for use with the CB tools in the WebSphere Application Server, *Enterprise Edition*, you can select only a single EJB group. You cannot select individual enterprise beans.
2. Click the right mouse button and then do *one* of the following:

   - To export both enterprise bean code and deployed classes to a deployed JAR file, select **Export > Deployed JAR** from the pop-up menu.
   - To export enterprise bean code without deployed classes to a generic EJB JAR file, select **Export > EJB JAR** from the pop-up menu.
   - To export enterprise bean code without deployed classes to an EJB JAR file intended for the CB tools in WebSphere Application Server, Enterprise Edition, select **Export > EJB JAR for CB** from the pop-up menu.

   The Export SmartGuide opens. (The title of the SmartGuide will differ depending on the menu item that you used to open the SmartGuide.)
3. In the **JAR file** field, type the name of the JAR file that you want to create.

   Alternatively, click the **Browse** button to select an existing JAR file to overwrite. The SmartGuide marks all of the class files associated with the selected EJB groups or individual enterprise beans for exportation to the JAR file.
4. Select the file types that you want to export.

   By default, all files of a selected type are exported. To see a list of the files that will be exported and to specify individual files, click the **Details** button next to a file type.
5. To ensure that any types or resources referenced by the selected EJB group or enterprise bean are also exported, click the **Select referenced types and resources** button.
6. Specify additional options as appropriate using the **Options** check boxes.

   - If you plan to debug the exported classes using an external debugger, select **Include debug attributes in .class files**. Debugging information will be included for the exported classes.
   - To compress the exported files, select **Compress the contents of the JAR file**.
   - If you are exporting to an existing JAR file and you do not want to be warned about overwriting it, select **Overwrite existing files without warning**.
   - If you are exporting to an EJB JAR file for Component Broker and you want to have the Component Broker import tools opened on the JAR file after exportation, select **Launch Component Broker**. (This check box only appears on the Export to an EJB JAR File for Component Broker SmartGuide.)
7. Click **Finish**.

After exportation
If you are deploying your enterprise beans to an application server other than the WebSphere Application Server, Advanced Edition, *and* your enterprise beans use either access beans or associations, you must ensure that the run-time JAR file required for access beans and associations is deployed to the application server. This is required even if you have a server-side situation where your enterprise bean uses an access bean to access another enterprise bean. Information about deploying the run-time JAR file is found in the topic "Deploying the run-time JAR file for access beans and associations."

If you exported to an EJB JAR file specifically for the CB deployment tools and you have the WebSphere Application Server, Enterprise Edition installed, the CB importation tools are automatically launched if you have the **Launch Component Broker** option selected. For information about using the CB importation tools, see the online documentation for the WebSphere Application Server, Enterprise Edition.

To deploy your generic EJB JAR file to a non-IBM EJB server after exportation, you will need to launch the deployment tools provided by the vendor of that server.

If you test-run your EJB client on the target system and you encounter a "Class not found" exception, check if it relates to one of the classes below. If it does, add the ejb_sstub.zip file to your client system and set your client system's class path accordingly. The ejb_sstub.zip file can be found in the *install_root*\eab\runtime35 directory. The classes packaged in the ejb_sstub.zip file are:

java.io._DataOutput_Stub
java.io._ObjectInput_Stub
java.io._DataInput_Stub
java.io._ObjectOutput_Stub
java.util._EventListener_Stub
java.lang._Cloneable_Stub

RELATED CONCEPTS

Overview of the EJB Development Environment

RELATED TASKS

Using the EJB Development Environment: overview
Deploying the run-time JAR file for access beans and associations
Exporting client-side code to client JAR files
Moving or copying enterprise beans between repositories

RELATED REFERENCES

Default type mappings for deployment to Component Broker
Finder helpers generated for deployment to Component Broker

Exporting client-side code to client JAR files

Once you have tested your enterprise beans, you can export the client-side code to a client JAR file and export your enterprise beans to an EJB JAR file or a deployed JAR file. (Information about exporting enterprise beans is found in the topic "Exporting enterprise beans to EJB or deployed JAR files".)

A client JAR file contains all the classes and interfaces used to access enterprise beans. After the JAR file is created, you can use it to execute the client application.

Notes

- Instead of adding the server project to the class path of the client project, we recommend that you include the client JAR file in the client class path before executing the client application. This provides better code isolation between the client and server code.
- When you test run your EJB client on the target system and you encounter a "Class not found" exception, you should check if it relates to one of the classes below. If it does, you should add the ejb_sstub.zip file to your client system and set your client system's class path accordingly. The ejb_sstub.zip file can be found in the *install_root*\eab\runtime35 directory. The classes packaged in the ejb_sstub.zip file are:

  java.io._DataOutput_Stub
  java.io._ObjectInput_Stub
  java.io._DataInput_Stub
  java.io._ObjectOutput_Stub
  java.util._EventListener_Stub
  java.lang._Cloneable_Stub

  Note that new enterprise beans and clients developed in VisualAge for Java, Version 3.5, do not require the ejb_sstub.zip file. However, if you developed clients using older versions of VisualAge for Java, you may still need the ejb_sstub.zip file.

Information about exporting enterprise beans into a repository is found in the topic "Moving or copying enterprise beans between repositories".

To export client-side code to a client JAR file:

1. In the Enterprise Beans pane of the EJB page, select one or more EJB groups or enterprise beans.
2. Click mouse button 2, then select **Export - Client JAR** from the pop-up menu.

   The Export to a Client JAR File SmartGuide appears. (Note that the title of the SmartGuide will differ depending on the menu item you used to open the SmartGuide, but the controls on the SmartGuide are exactly the same regardless of the menu item you selected.)
3. In the **JAR file** field, type the name of the JAR file you want to create.

   Alternatively, click the **Browse** button to select an existing JAR file to overwrite. The SmartGuide marks all of the class files associated with the selected EJB groups or individual enterprise beans for exportation to the JAR file.
4. Select the individual file types that you want to export by clicking one or more of the following check boxes:

   - **beans** Export all the beans in the EJB group or the individual enterprise bean you selected.
   - **.class** Export all the bytecode files for the selected EJB group or enterprise bean.
   - **.java** Export all the source code files for the selected EJB group or enterprise bean.

- **resource** Export all the non-.java and non-.class files for the selected EJB group or enterprise bean.
- **Details** By default, all the files for the selected file type are exported. Click the **Details** button next to a file type to see a list of the files that will be exported, and to specify individual files to export.

5. To ensure that any types or resources referenced by the selected EJB group or enterprise bean are also exported, click the **Select referenced types and resources** button. (You may also want to browse the selected reference types and deselect any types or libraries that you know are already part of the client-side deployment library for your application server.)
6. Specify additional options as appropriate using the **Options** check boxes as follows:

   - **Include debug attributes in .class files** If you want to debug the exported classes using an external debugger, click this check box to include debugging information for the exported classes.
   - **Compress the contents of the JAR file** Compress the exported files.
   - **Overwrite existing files without warning** If you are exporting to an existing JAR file and you do not want to be warned about overwriting it, click this check box.

7. Click **Finish.**

RELATED CONCEPTS

Overview of the EJB Development Environment

RELATED TASKS

Using the EJB Development Environment: overview
Exporting enterprise beans to EJB or deployed JAR files
Moving or copying enterprise beans between repositories

Deploying the run-time JAR file for access beans and associations

If your enterprise beans employ either access beans or associations *and* you are deploying your enterprise beans on an application server other than the WebSphere Application Server, Advanced Edition, you will also need to deploy the run-time JAR file and include it in the application server's class path. This run-time JAR file contains both access bean and association run-time code that is required to support the operation of your access beans and associations.

To deploy the run-time JAR file:

1. Navigate to the following directory (where *path* is the VisualAge for Java install path):

   path\eab\runtime35
2. Copy the file ivjejb35.jar to an appropriate directory on your application server.
3. Include the ivjejb35.jar file in the class path of the application server.
4. If your client application is running outside of your application server, then you also need to include the JAR file in the class path of the client application.

RELATED CONCEPTS

Overview of the EJB Development Environment

RELATED TASKS

Using the EJB Development Environment: overview
Exporting enterprise beans to EJB or deployed JAR files
Exporting client-side code to client JAR files
Moving or copying enterprise beans between repositories

Chapter 11. Working Around Problems

Handling EJB serialization problems

You may encounter a serial version UID mismatch exception when running an EJB client inside VisualAge for Java and accessing an enterprise bean contained in a server that is running outside of VisualAge for Java. This may also occur when you are running an EJB client outside of VisualAge for Java and accessing an enterprise bean that is running in the EJB test environment inside VisualAge for Java. The problem is caused by a difference in some of the compilation techniques used in VisualAge for Java versus the standard Java compiler. To understand more about the cause of this problem, see the paper entitled "Handling EJB Serialization Problems in VisualAge for Java", which is available from the VisualAge Developer Domain (VADD) Web site at the following URL:

www.ibm.com/software/vadd

To work around the problem, do one of the following:

- If you must have the source code loaded in the IDE so that you can edit and debug it, then specify a valid UID in the class that is causing the problem. Be sure to re-deploy the enterprise bean to the server if you have changed any EJB-related class definitions.
- If you are satisfied that you will not need to edit the code that is causing the problem, then you can workaround the problem by having only a .class version of the code loaded into the IDE. Save your source code version in the repository, then export the .class version to the file system and import that version into the workspace.

RELATED CONCEPTS

Overview of the EJB Development Environment

RELATED TASKS

Using the EJB Development Environment: overview

Handling problems caused by DB2 connection or application limits

If you are developing enterprise beans, you may be unable to start the Persistent Name Server because the maximum number of database connections or applications has been exceeded. In this case, the connections or applications for a previously active EJB server have probably not been released. The message in the Console window will be:

```
Failure while creating connection COM.ibm.db2.jdbc.app.DB2Exception:
[IBM][CLI Driver] SQL1224N A database agent could not be started
to service a request, or was terminated as a result of a database
system shutdown or a force command.
SQLSTATE=55032
```

You need to find and terminate the connections or applications that have not been closed. This can be done by issuing the following commands from a DB2 command window:

1. List the active connections by issuing a `DB2 LIST APPLICATIONS` command.

© Copyright IBM Corp. 1998, 2000

2. Either terminate the connections by closing the VisualAge for Java IDE and then restarting it, or manually terminate the connections by issuing one of the following two commands:

   - Use the `DB2 FORCE APPLICATION ALL` command to terminate all DB2 connections.
   - Use the `DB2 FORCE APPLICATION` command to terminate selected DB2 connections. For example, issue the following command to terminate application handles 3, 45 and 87 (as determined by using the `LIST APPLICATIONS` command):

     ```
     DB2 FORCE APPLICATION( 3,45 87)
     ```

RELATED CONCEPTS

Limitations

RELATED TASKS

Creating an EJB server configuration
Starting EJB servers
Stopping EJB servers

Recovering and reinstalling workspaces

If your workspace becomes corrupt, you will need to recover or reinstall the workspace before you can continue working in the EJB Development Environment.

RELATED CONCEPTS

Overview of the EJB Development Environment

RELATED TASKS

Using the EJB Development Environment: overview
Reinstalling the EJB Development Environment
Recovering EJB groups and enterprise beans

Reinstalling the EJB Development Environment

If your workspace becomes corrupt, you will need to recover or reinstall the workspace and then reinstall the EJB Development Environment feature. Information about installing the features required for the EJB Development Environment is found in the topic “Loading the required features”.

RELATED CONCEPTS

Overview of the EJB Development Environment

RELATED TASKS

Using the EJB Development Environment: overview
Loading the required features
Recovering and reinstalling workspaces
Recovering EJB groups and enterprise beans

Recovering EJB groups and enterprise beans

In the event that the workspace is corrupted or the VisualAge for Java application terminates abnormally, you will need to recover your EJB groups and enterprise beans from the repository once you have restored the workspace and reinstalled the EJB Development Environment feature. Information about adding EJB groups from repositories is found in the topic "Adding EJB groups". Information about adding enterprise beans is found in the topic "Adding enterprise beans".

If an EJB group or enterprise bean on the EJB page is corrupted, you can delete the project associated with the EJB group or enterprise bean and reload the project. An EJB group can become corrupted if you modify the reserved package of the EJB group manually.

RELATED CONCEPTS

Overview of the EJB Development Environment

RELATED TASKS

Using the EJB Development Environment: overview
Adding EJB groups
Adding enterprise beans
Recovering and reinstalling workspaces
Reinstalling the EJB Development Environment

Chapter 12. Tutorials

Preparing for the tutorials

The EJB Development Environment tutorials introduce you to the many tools of the EJB Development Environment. Specifically, the tutorials show you how you can use the EJB Development Environment to develop session and entity enterprise beans. The enterprise beans used in the tutorials contain simple business methods that you access from a client. The tutorials illustrate how enterprise beans must be structured to operate correctly on an EJB server, such as the WebSphere Application Server.

Description of the tutorials
There are two tutorials:

Increment
In the Increment tutorial, you use the EJB Development Environment to develop and test a CMP entity bean that increments a counter.

Hello World!
In the Hello World! tutorial, you use the Visual Composition Editor to develop a simple client application using an access bean and a session bean created in the EJB Development Environment.

Before you begin the tutorials
Before you can work with the tutorials, you must ensure that VisualAge for Java and DB2 are correctly installed and configured. You must also have loaded the IBM EJB Development Environment feature and have added a JDBC driver to the class path. You can find information about loading the required features in the topic "Loading the required features". You can find information about adding a JDBC driver to the class path in the topic "Adding a JDBC driver to the class path".

Once you have loaded the required features and added a JDBC driver to the class path, you can prepare for the tutorials. The following sections show you how to:

- Create a new project and package.
- Copy the required classes and interfaces to the new project.
- Start the DB2 servers and create the DB2 sample database.

Create a new project and package
To create a new project and package:

1. In the VisualAge for Java Workbench Projects page, from the **Selected** menu, choose **Add** > **Project**. The Add Project SmartGuide appears.
2. In the **Create a new project named** field, type `MyEJBProject` and click **Finish**. The new project MyEJBProject appears in the All Projects pane.
3. Select MyEJBProject, then from the **Selected** menu, choose **Add** > **Package**. The Add Package SmartGuide appears.
4. In the **Create a new package named** field, type `myejbpackage` and click **Finish**. The new package myejbpackage appears under MyEJBProject.

Copy the required classes and interfaces to the new project
To copy the required classes and interfaces to the new project:

© Copyright IBM Corp. 1998, 2000

1. Expand the IBM EJB Samples project and expand the com.ibm.ivj.ejb.samples.increment package, then select the following classes and interfaces:

   ```
   Increment
   IncrementBean
   IncrementClient
   IncrementHome
   IncrementKey
   ```

2. Click mouse button 2, then select **Reorganize** > **Copy** from the pop-up. The Copying Types dialog appears.
3. Deselect the **Rename the copy** check box.
4. Beside the **Copy your items to Package** field, click **Browse** and type `myejbpackage` in the **Pattern** field, then click **OK**.
5. Click **OK**. If a Warning dialog appears, click **OK**. (Note that if an **X** error icon appears beside the copied IncrementClient class, it is expected. It will disappear when you later generate deployed code in the Increment tutorial.)

Start the DB2 servers and create the sample databases

To start the DB2 servers and create the sample databases:

1. On the Windows desktop, select **Start** > **Programs** > **DB2 for Windows** > **Command Window** to open a DB2 command window. On the command line, issue the following two commands to ensure that the DB2 database server and the DB2 administration server are running:

   ```
   db2start
   db2admin start
   ```

   If the servers are already running, you are advised of this by a message. (Note that depending on how you have configured DB2, you may need to start the DB2 servers any time you reboot your machine in order to work with the tutorials.)
2. In the DB2 command window, navigate to the IBM EJB Samples directory (where *path* is the install path of VisualAge for Java). For example:

   path\ide\project_resources\IBM EJB Samples
3. In the IBM EJB Samples directory, issue the following command to create the DB2 database required for the samples:

   ```
   db2 -svtf sampleDB.clp
   ```

4. Close the DB2 command window.

Once you have completed these instructions, you can complete the tutorials in any order.

RELATED TASKS

Loading the required features
Adding a JDBC driver to the classpath
Tutorial: Hello World
Tutorial: Increment

Tutorial: Increment

Objective
This tutorial introduces you to the EJB Development Environment, which allows you to accomplish numerous EJB development activities, such as developing and testing enterprise beans. Specifically, the tutorial shows you how the EJB Development Environment is used to develop a container-managed persistence (CMP) entity enterprise bean. This bean contains a simple business method that you access from a simple client application. The tutorial illustrates how an enterprise bean must be structured to operate correctly on an EJB server.

Time Required
Approximately 45 - 60 minutes.

Before You Begin
Before you can work with this tutorial, you must ensure that you have completed the prerequisite tasks in the topic "Preparing for the Tutorials".

Description
In this tutorial, you use the EJB Development Environment to develop and test a CMP entity enterprise bean named Increment that increments a counter. The following classes and interfaces are provided with source:

- Increment
- IncrementHome
- IncrementBean
- IncrementClient
- IncrementKey

The Increment tutorial is divided into two independent parts:

- Part 1 shows you how to quickly run the ready-made Increment sample. (You can also examine the ready-made code of the Increment sample to learn more about the structure of enterprise beans.)
- Part 2 shows you how to build the Increment sample for yourself from scratch.

Part 1 - Running the ready-made Increment sample
In this part of the tutorial, you run the ready-made Increment sample.

Create an EJB server configuration and start the servers:

1. In the Workbench Projects page, click on the **EJB** tab. The EJB page appears.
2. In the Enterprise Beans pane, select IBMEJBSamples, then click mouse button 2 and select **Add To** > **Server Configuration**. The EJB Server Configuration browser appears:
3. In the Servers pane, select **EJB Server (server1)**.
4. Click mouse button 2 and select **Properties** from the pop-up menu. The Properties for EJB Server dialog appears.
5. In the **Data Source** field, ensure that the following URL is specified.

   ```
   jdbc:db2:sampleDB
   ```

6. Close the Properties dialog.
7. In the Servers pane, select **EJB Server (server1)** and click mouse button 2, then select **Create Database Table** from the pop-up menu.
8. From the **Workspace** menu, select **Tools > WebSphere Test Environment**. The WebSphere Test Environment Control Center appears.

9. In the left pane of the Control Centre, click **Persistent Name Server**, then in the right pane, click **Start Name Server**. The Console window appears in the background and monitors the server. (Detailed information about the Persistent Name Server is found in the online help documentation for the WebSphere Test Environment component.)
10. In the status area of the WebSphere Test Environment Control Center, ensure that the message "Persistent name server is started" appears.
11. In the EJB Server Configuration browser, select **EJB Server (server1)** in the Servers pane, then click mouse button 2 and select **Start Server**. The Console window now also monitors the EJB server.

    (If you have problems starting the EJB server, in the Console window, select the EJB server process and examine the first error message that appears in the Output pane. If the error message indicates that the maximum number of database connections or applications has been exceeded, refer to the topic "Handling Problems Caused by DB2 Connection or Application Limits" for information on how to correct this problem. If, however, an error message indicates that the JDBC driver cannot be loaded, or if the debugger has opened, you should try creating a new project and importing the DB2 JDBC driver into the new project. In some situations, importing the JDBC driver may be necessary when working with entity enterprise beans. Note that once you have addressed the problem that caused the EJB server to fail to start, you should stop the Persistent Name Server, start it again, and then start the EJB server.)
12. In the Console, check that the EJB Server is ready to process a call from the client by selecting the EJB Server entry at the bottom of the All Programs pane and then looking for the following message at the bottom of the Output pane:

    ```
    Server open for business..
    ```

    (The message should appear in the last line at the bottom of the output text. However, it may take a moment or two to appear.)

Set up and run the Increment client application:

1. In the Workbench, click on the **Projects** tab to go to the Projects page.
2. Expand the IBM EJB Samples project, then expand the following package:

   com.ibm.ivj.ejb.samples.increment
3. Select the IncrementClient class, then click mouse button 2 and select **Properties**. The Properties dialog appears.
4. Click the **Class Path** tab.
5. Beside the **Project Path** field, click the **Edit** button. The Class Path dialog appears.
6. Ensure that the **IBM WebSphere Test Environment** project and the **IBM Enterprise Extension Libraries** are selected, then click **OK**.
7. Ensure that the **Save in repository (as default)** check box is selected, then click **OK**.
8. Select the IncrementClient class, click mouse button 2 and select **Run** > **Run main**.
9. In the All Programs pane of the Console, select the following entry:

   ```
   com.ibm.ivj.ejb.samples.increment.IncrementClient.main()
   ```

10. In the Output pane of the Console, the client has generated the following output:

```
Obtaining initial context.
Looking up Increment Enterprise Bean.
Obtaining IncrementHome object.
Looking for increment object named: TEST
The object was not found. I will create it.
Object named: TEST has count: 0
Now, object named: TEST has count: 1
Now, object named: TEST has count: 2
Now, object named: TEST has count: 3
Now, object named: TEST has count: 4
Now, object named: TEST has count: 5
```

(Note that the output values will be different if you ran the client earlier.)

11. In the Workbench Projects page, select the IncrementClient class again and select **Run** > **Run main**.
12. In the All Programs pane of the Console, select the following entry:

```
com.ibm.ivj.ejb.samples.increment.IncrementClient.main()
```

13. In the Output pane of the Console, the client has generated the following output:

```
Obtaining initial context.
Looking up Increment Enterprise Bean.
Obtaining IncrementHome object.
Looking for increment object named: TEST
Object named: TEST has count: 5
Now, object named: TEST has count: 6
Now, object named: TEST has count: 7
Now, object named: TEST has count: 8
Now, object named: TEST has count: 9
Now, object named: TEST has count: 10
```

(Note that the output values will be different if you ran the client earlier.)

Stop the EJB server and remove the EJB server configuration:

1. In the EJB Server Configuration browser, select EJB server (server1) in the Servers pane, then click mouse button 2 and select **Stop Server**.
2. Select EJB Server (server1) again, then click mouse button 2 and select **Remove Server**.

Part 2 - Building the Increment sample for yourself

In this part of the tutorial, you build the Increment sample for yourself from scratch.

Add an EJB group and the Increment enterprise bean:

1. On the EJB page, from the **EJB** menu select **Add** > **EJB Group**. The Add EJB Group SmartGuide appears.
2. Beside the **Project** field, click the **Browse** button and select MyEJBProject from the list of projects, then click **OK**.

   Alternatively, you can type `MyEJBProject` directly in the **Project** field.
3. In the **Create a new EJB group named** field, type in `MyIncGroup`.

4. Click **Finish**. The group MyIncGroup is created and is displayed in the Enterprise Beans pane.
5. In the Enterprise Beans pane, select MyIncGroup, then click mouse button 2 and select **Add** > **Enterprise Bean** from the pop-up menu. The Create Enterprise Bean SmartGuide appears.
6. In the **Bean name** field, type `Increment`.
7. Select the **Use an existing Bean class** radio button.
8. Click **Finish**. (The EJB Development Environment tools introspect the existing bean class to create the appropriate bean type, such as a session bean, a CMP entity bean, or a BMP entity bean.) The Increment enterprise bean appears under MyIncGroup.

Define your CMP and key fields:

1. In the Enterprise Beans pane, select the Increment enterprise bean.
2. In the upper right corner of the Types pane, click the **F** icon to toggle to the Properties pane.
3. In the Properties pane, select `primaryKey`, then click mouse button 2 and ensure that a check mark appears beside **Key Field**.
4. In the Properties pane, select `count`, then click mouse button 2 and select **Container Managed**.

Create the database table:

1. On the Windows desktop, select **Start** > **Programs** > **DB2 for Windows** > **Command Window** to open a DB2 command window.
2. In the DB2 command window, navigate to the following IBM EJB Samples directory (where *path* is the install path of VisualAge for Java):

   path\ide\project_resources\IBM EJB Samples
3. In the IBM EJB Samples directory, issue the following command to create the DB2 database table required for the samples:

   ```
   db2 -svtf increment.clp
   ```
4. Close the DB2 command window.

Define your database schemas:

1. In the Enterprise Beans pane, select MyIncGroup.
2. From the **EJB** menu, select **Open To** > **Database Schemas**. The Schema Browser appears.
3. From the **Schemas** menu, select **Import/Export Schema** > **Import Schema from Database**. The Information Required dialog appears.
4. In the **Enter the name for the schema** field, type `Increments` and click **OK**. The Database Connection Info dialog appears.
5. In the **Connection Type** field, type the following driver name:

   ```
   COM.ibm.db2.jdbc.app.DB2Driver
   ```
6. In the **Data Source** list box, type the following entry:

   ```
   jdbc:db2:sampleDB
   ```

   This is the name of the database that you created using the sampleDB.clp script in the “Preparing for the Tutorials” topic.

7. Click **OK**. The Select Tables dialog appears.
8. Click the **Build Table List** button.
9. In the **Name** column, select INCREMENTS.
10. Click **OK**. The imported database schema appears in the Schemas pane of the Schema browser.
11. Close the Schema browser.

Map your database schemas:

1. In the EJB page, from the **EJB** menu, select **Open To** > **Schema Maps**. The Map browser appears.
2. From the **Datastore_Maps** menu, select **New EJB Group Map**. The New Datastore Map dialog appears.
3. In the **Name** field, type `EJBMap`.
4. In the **EJB Group** field, select `MyIncGroup`.
5. In the **Schema** field, select `Increments` and click **OK**. `EJBMap` appears in the Datastore Maps pane of the Map browser.
6. In the Datastore Maps pane, select `EJBMap`.
7. In the Persistent Classes pane, select `Increment`.
8. Position the cursor in the Table Maps pane and click mouse button 2, then select **New Table Map** > **Add Table Map with no inheritance**. The **Table Map with No Inheritance** dialog appears.
9. In the **Table** field, select `INCREMENTS` and click **OK**.
10. In the Table Maps pane, select INCREMENTS (primary) and click mouse button 2, then select **Edit Property Maps**. The Property Map Editor appears.
11. In the Class Attribute column, select `primary Key`.
12. Immediately to the right, in the Map Type column, select `Simple`, and to the right in the Table Column, select PRIMARYKEY.
13. In the Class Attribute column, select `count`.
14. Immediately to the right, in the Map Type column, select `Simple`, and to the right in the Table Column, select COUNT.
15. Click **OK**. The mapping is complete.
16. Close the Map browser.

Generated the deployed code:

1. In the Enterprise Beans pane of the EJB page, select MyIncGroup.
2. From the **EJB** menu, select **Generate Deployed Code**.

Create an EJB server configuration and start the servers:

1. In the Enterprise Beans pane, select MyIncGroup, then click mouse button 2 and select **Add to** > **Server Configuration**. The EJB Server Configuration browser appears.
2. In the Servers pane, select **EJB Server (server1)**.
3. Click mouse button 2 and select **Properties** from the pop-up menu. The Properties for EJB Server dialog appears.
4. In the **Data Source** field, ensure that the following URL is specified.

   ```
   jdbc:db2:sampleDB
   ```

5. Close the Properties dialog.
6. From the **Workspace** menu, select **Tools > WebSphere Test Environment**. The WebSphere Test Environment Control Center appears.

7. In the left pane of the Control Centre, click **Persistent Name Server**, then in the right pane, click **Start Name Server**. The Console window appears in the background and monitors the server. (Detailed information about the Persistent Name Server is found in the online help documentation for the WebSphere Test Environment component.)
8. In the status area of the WebSphere Test Environment Control Center, ensure that the message "Persistent name server is started" appears.
9. In the EJB Server Configuration browser, select **EJB Server (server1)** in the Servers pane, then click mouse button 2 and select **Start Server**. The Console window now also monitors the EJB server.

   (If you have problems starting the EJB server, in the Console window, select the EJB server process and examine the first error message that appears in the Output pane. If the error message indicates that the maximum number of database connections or applications has been exceeded, refer to the topic "Handling Problems Caused by DB2 Connection or Application Limits" for information on how to correct this problem. If, however, an error message indicates that the JDBC driver cannot be loaded, or if the debugger has opened, you should try creating a new project and importing the DB2 JDBC driver into the new project. In some situations, importing the JDBC driver may be necessary when working with entity enterprise beans. Note that once you have addressed the problem that caused the EJB server to fail to start, you should stop the two name servers, start them again, and then start the EJB server.)
10. In the Console, check that the EJB Server is ready to process a call from the client by selecting the EJB Server entry at the bottom of the All Programs pane and then looking for the following message at the bottom of the Output pane:

    ```
    Server open for business..
    ```
    (The message should appear in the last line at the bottom of the output text. However, it may take a moment or two to appear.)

Run the EJB test client:

1. In the Enterprise Beans pane of the EJB Server Configuration browser, expand MyIncGroup and select the Increment enterprise bean, then click mouse button 2 and select **Run Test Client**. The EJB Test Client appears.
2. Click **Lookup**. This directs the test client to construct an initial naming context and query the name server for the JNDI name selected in the JNDI list box. When the connection is established to the name server and an instance of the home interface is retrieved from the name server, the Home interface page for the enterprise bean appears.
3. In the Methods pane of the Home interface page, select the create(IncrementKey) method.
4. In the Details pane of the Home interface page, type `104` in the text area beside String primaryKey.
5. In the Methods pane of the Home interface page, select the create(IncrementKey) method again, then click the **Invoke** button to run the method. The Remote interface page appears.
6. In the Methods pane of the Remote interface page, select the increment() method and click the **Invoke** button. The counter is incremented by 1 and the number 1 appears in the **Details** pane.
7. Select the increment() method again and click the **Invoke** button again to increment the result to 2.

8. In the Primary key pane of the Remote interface page, select String primaryKey and then right-click and select **Inspect**. The Inspector window appears. The Inspector window allows you to inspect any object that is returned from a method call, such as complex types that are non-primitive.
9. Close the Inspector window and the test client window.

Set up and run the Increment client application:

1. In the Workbench, click on the **Projects** tab to go to the Projects page.
2. Expand the MyEJBProject project, then expand the following package:

   myejbpackage
3. Select the IncrementClient class, then click mouse button 2 and select **Properties**. The Properties dialog appears.
4. Click the **Class Path** tab.
5. Beside the **Project Path** field, click the **Edit** button. The Class Path dialog appears.
6. Ensure that the **IBM WebSphere Test Environment** project and the **IBM Enterprise Extension Libraries** are selected, then click **OK**.
7. Ensure that the **Save in repository (as default)** check box is selected, then click **OK**.
8. Select the IncrementClient class, then click mouse button 2 and select **Run** > **Run main**.
9. In the All Programs pane of the Console, select the following entry:

   ```
   myejbpackage.IncrementClient.main()
   ```
10. In the Output pane of the Console, the client has generated the following output:

    ```
    Obtaining initial context.
    Looking up Increment Enterprise Bean.
    Obtaining IncrementHome object.
    Looking for increment object named: TEST
    Object named: TEST has count: 10
    Now, object named: TEST has count: 11
    Now, object named: TEST has count: 12
    Now, object named: TEST has count: 13
    Now, object named: TEST has count: 14
    Now, object named: TEST has count: 15
    ```

    (Note that the output values will be different if you did not run the client earlier in Part 1 of this tutorial.)
11. In the Workbench Projects page, select the IncrementClient class again and select **Run** > **Run main**.
12. In the All Programs pane of the Console, select the following entry:

    ```
    myejbpackage.IncrementClient.main()
    ```
13. In the Output pane, the counter is incremented by another five digits:

    ```
    Obtaining initial context.
    Looking up Increment Enterprise Bean.
    Obtaining IncrementHome object.
    Looking for increment object named: TEST
    Object named: TEST has count: 15
    Now, object named: TEST has count: 16
    ```

```
Now, object named: TEST has count: 17
Now, object named: TEST has count: 18
Now, object named: TEST has count: 19
Now, object named: TEST has count: 20
```

(Note that the output values will be different if you did not run the client earlier in Part 1 of this tutorial.)

Stop the EJB server and remove the EJB server configuration:

1. In the EJB Server Configuration browser, select EJB server (server1) in the Servers pane, then click mouse button 2 and select **Stop Server**.
2. Select EJB Server (server1) again, then click mouse button 2 and select **Remove Server**.
3. Close the EJB Server Configuration browser.

Conclusion
In this tutorial, you have learned how the EJB Development Environment is used to develop a container-managed persistence (CMP) entity enterprise bean. The tutorial illustrated how an enterprise bean must be structured to operate correctly on the WebSphere EJB server.

RELATED TASKS

Preparing for the Tutorials
Tutorial: Hello World!
Handling Problems Caused by DB2 Connection or Application Limits

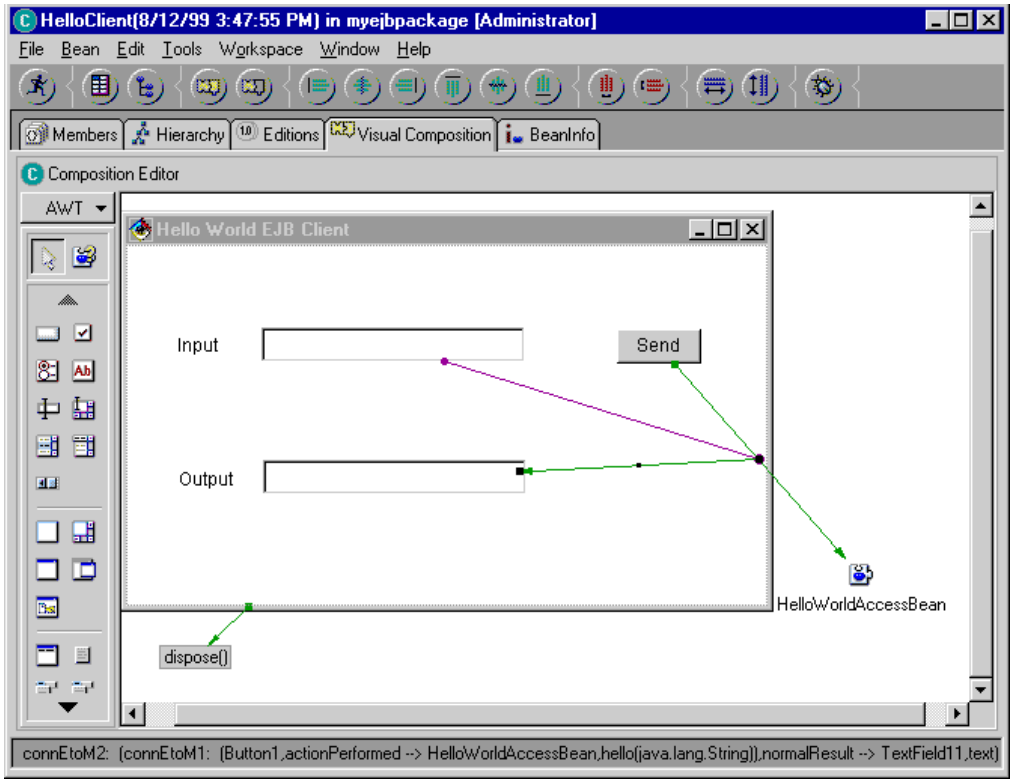
Tutorial: Hello World!

Objective
This tutorial introduces you to the EJB Development Environment, which allows you to accomplish numerous EJB development activities, such as developing and testing enterprise beans. Specifically, the tutorial shows you how to use the Visual Composition Editor to develop a simple client application using an access bean and a session bean created in the EJB Development Environment.

Time Required
Approximately 60 minutes.

Before You Begin
Before you can work with this tutorial, you must ensure that you have completed the prerequisite tasks in the topic “Preparing for the Tutorials”.

Description
In this tutorial, you use the EJB Development Environment to develop and test a session enterprise bean and an access bean. You then use the Visual Composition Editor to develop a client application that accesses the enterprise bean. When you have finished building the client in the Visual Composition Editor, it should look like the following figure:



The following classes are provided with source:

HelloWorld
HelloWorldBean
HelloWorldAccessBean
HelloWorldHome
HelloClient

The Hello World tutorial is divided into two independent parts:

- Part 1 shows you how to quickly run the ready-made Hello World sample. (You can also examine the ready-made code of the Hello World sample to learn more about the structure of enterprise beans.)
- Part 2 shows you how to build the Hello World sample for yourself from scratch.

Part 1 - Running the ready-made Hello World sample
In this part of the tutorial, you run the ready-made Hello World sample.

Create an EJB server configuration and start the servers:

1. In the Workbench Projects page, click on the **EJB** tab. The EJB page appears.
2. In the Enterprise Beans pane, select the EJB group IBMEJBSamples, then click mouse button 2 and select **Add To > Server Configuration**. The EJB Server Configuration browser appears.
3. From the **Workspace** menu, select **Tools > WebSphere Test Environment**. The WebSphere Test Environment Control Centre appears.
4. In the left pane of the Control Centre, click **Persistent Name Server**, then in the right pane, click **Start Name Server**. The Console window appears in the

background and monitors the server. (Detailed information about the Persistent Name Server is found in the online help documentation for the WebSphere Test Environment component.)

5. In the status area of the WebSphere Test Environment Control Center, ensure that the message "Persistent name server is started" appears.
6. In the EJB Server Configuration browser, select **EJB Server (server1)** in the Servers pane, then click mouse button 2 and select **Start Server**. The Console window now also monitors the EJB server.

   (If you have problems starting the EJB server, in the Console window, select the EJB server process and examine the first error message that appears in the Output pane. If the error message indicates that the maximum number of database connections or applications has been exceeded, refer to the topic "Handling Problems Caused by DB2 Connection or Application Limits" for information on how to correct this problem. If, however, an error message indicates that the JDBC driver cannot be loaded, or if the debugger has opened, you should try creating a new project and importing the DB2 JDBC driver into the new project. In some situations, importing the JDBC driver may be necessary when working with entity enterprise beans. Note that once you have addressed the problem that caused the EJB server to fail to start, you should stop Persistent Name Server, start it again, and then start the EJB server.)
7. In the Console, check that the EJB Server is ready to process a call from the client by selecting the EJB Server entry at the bottom of the All Programs pane and then looking for the following message in the Output pane:

   ```
   Server open for business...
   ```

   (The message should appear in the last line at the bottom of the output text. However, it may take a moment or two to appear.)

Set up and run the Hello World client application:

1. In the Workbench, click on the Projects tab to go the Projects page.
2. Expand the project IBM EJB Samples and then expand the package com.ibm.ivj.ejb.samples.helloworld.
3. Select the HelloClient class, then click mouse button 2 and select **Properties**. The Properties dialog appears.
4. Click the Class Path tab.
5. Beside the **Project Path** field, click the **Edit** button. The Class Path dialog appears.
6. Ensure that the **IBM WebSphere Test Environment** project is selected, then click **OK**.
7. At the bottom of the dialog, ensure that the **Save in repository (as default)** check box is selected, then click **OK**.
8. Select the HelloClient class, then click mouse button 2 and select **Run > Run main**. The Hello World EJB Client appears.
9. At the bottom of the All Programs pane in the Console window, select the EJB server process.
10. Position the Console window to the right of the screen.
11. Select the Hello World EJB Client window and position it to the left of screen.
12. In the Hello World EJB Client, in the **Input** field, type `HelloWorld` and click the **Send** button.
13. In the **Output** field, the following string appears:

```
You said: HelloWorld
```

14. Close the Hello World EJB Client.

Stop the EJB server and remove the EJB server configuration:

1. In the EJB Server Configuration browser, select the EJB server (server1) in the Servers pane, then click mouse button 2 and select **Stop Server** from the pop-up menu.
2. Select the EJB Server (server1) again, then click mouse button 2 and select **Remove Server**.

Part 2 - Building the Hello World sample for yourself
In this part of the tutorial, you build the Hello World sample for yourself from scratch.

Add an EJB group and the HelloWorld enterprise bean:

1. In the Workbench Projects page, click the **EJB** tab. The EJB page appears.
2. From the **EJB** menu, select **Add** > **EJB Group**. The Add EJB Group SmartGuide appears.
3. Beside the **Project** field, click the **Browse** button and select MyEJBProject from the list of projects, then click **OK**.

   Alternatively, you can type `MyEJBProject` directly in the **Project** field.
4. In the **Create a new EJB group named** field, type `MyHelloGroup`.
5. Click **Finish** to create the group. The group MyHelloGroup is displayed in the Enterprise Beans pane.
6. In the Enterprise Beans pane, select MyHelloGroup, then click mouse button 2 and select **Add** > **Enterprise Bean** from the pop-up menu. The Create Enterprise Bean SmartGuide appears.
7. In the **Bean name** field, type `HelloWorld`.
8. In the **Project** field, ensure that `MyEJBProject` appears.
9. In the **Package** field, ensure that `myejbpackage` appears.
10. In the **Class** field, ensure that `HelloWorldBean` appears.
11. Click **Finish**. The HelloWorld enterprise bean appears under the group MyHelloGroup in the Enterprise Beans pane.

Add an hello() method:

1. In the Types pane, select HelloWorldBean and click mouse button 2, then select **Add** > **Method** from the pop-up menu. The Create Method SmartGuide appears.
2. In the text field, type `public String hello(String sayThis)` and click **Finish**. The new hello(String) method appears in the Members pane.
3. In the Source pane, modify the return statement to read as follows:

```
return "You said: " + sayThis;
```

4. In the Source pane, click mouse button 2 and select **Save**.
5. In the Members pane, select the hello(String) method and click mouse button 2, then select **Add To** > **EJB Remote Interface**.

Create an access bean and generate the deployed code:

1. In the Enterprise Beans pane, ensure the HelloWorld enterprise bean is selected.

2. From the **EJB** menu, select **Add > Access Bean**. The Create Access Bean SmartGuide appears.
3. In the SmartGuide, the required information already appears in the fields, so click **Finish** to generate the access bean.
4. Under MyHelloGroup in the Enterprise Beans pane, ensure that the HelloWorld enterprise bean is selected, then from the **EJB** menu, select **Generate Deployed Code**.

Create an HelloClient class:

1. Switch to the Projects page and expand the project MyEJBProject.
2. Select the myejbpackage package and click mouse button 2, then select **Add** > **Class**. The Create Class SmartGuide appears.
3. In the **Class Name** field, type HelloClient.
4. In the **Superclass** field, type `java.awt.Frame`.
5. Ensure that the **Compose the class visually** check box is selected, then click **Finish**. The Visual Composition Editor appears. On the free-form surface, a default window and the default dispose() method are displayed.

Use the Visual Composition Editor to create your client application:

1. In the Visual Composition Editor, double-click the title bar of the window in the free-form surface. The Properties dialog opens.
2. In the left column, click the **title** field, then in the right column, type `Hello World EJB Client` and close the dialog. The title appears in the title bar of the window.
3. On the beans palette, select the Choose Bean icon. The Choose Bean dialog appears.
4. Click the **Browse** button, then type `HelloWorld` in the **Pattern** field.
5. In the **Class Names** field, select HelloWorldAccessBean.
6. In the **Package Names** field, select myejbpackage and click **OK**.
7. In the **Name** field, type `HelloWorldAccessBean` and click **OK**.
8. Drop the bean in the lower-right corner of the free-form surface outside the window:
9. On the beans palette, select the Label bean and drop it in the upper left area of the client window.
10. Double-click the Label1 bean to open the Properties dialog, then in the text field, type `Input` and close the dialog.
11. On the beans palette, select the TextField bean and drop the bean to the right of the Input bean, then drag the ends of the TextField bean to expand it.
12. Select the Input bean, then from the **Edit** menu, select **Copy**.
13. From the **Edit** menu, select **Paste**. Drop the copied Input bean below the original Input bean.
14. Select the text field, then from the **Edit** menu, select **Copy**.
15. From the **Edit** menu, select **Paste**. Drop the copied text field to the right of the Input bean.
16. Double-click the new Input bean you copied to open the Properties dialog, then in the **text** field, type `Output` and close the dialog.
17. On the beans palette, select the Button bean and drop it to the right of the upper text field.
18. Double-click the new button to open the Properties dialog, then in the **label** field, type `Send` and close the dialog.

19. Select the **Send** button, click mouse button 2 and select **Connect** > **actionPerformed** from the pop-up menu, then position your cursor on the HelloWorldAccessBean object and click mouse button 2.
20. From the pop-up menu, select **Connectable Features**. The End Connection dialog appears.
21. Ensure that the **Method** radio button is selected.
22. In the Display pane, select hello(java.lang.String) and click **OK**.
23. Select the **Input** text field, then click mouse button 2 and select **Connect** > **text** from the pop-up menu.
24. Select the connection between the Send button and the HelloWorld object, then click mouse button 2 and select **arg0** from the pop-up menu.
25. Select the connection between the Send button and the HelloWorldAccessBean object, then click mouse button 2 and select **Connect > normalResult**.
26. Postion the cursor on the Output field, then click mouse button 2 and select **text**.
27. From the **Bean** menu, select **Save Bean**.

Create an EJB server configuration and start the servers:

1. Switch to the EJB page in the Workbench.
2. In the Enterprise Beans pane, select the group MyHelloGroup, then click mouse button 2 and select **Add To > Server Configuration**. The EJB Server Configuration browser appears.
3. From the **Workspace** menu, select **Tools > WebSphere Test Environment**. The WebSphere Test Environment Control Centre appears.
4. In the left pane of the Control Centre, click **Persistent Name Server**, then in the right pane, click **Start Name Server**. The Console window appears in the background and monitors the server. (Detailed information about the Persistent Name Server is found in the online help documentation for the WebSphere Test Environment component.)
5. In the status area of the WebSphere Test Environment Control Center, ensure that the message “Persistent name server is started” appears.
6. In the EJB Server Configuration browser, select **EJB Server (server1)** in the Servers pane, then click mouse button 2 and select **Start Server**. The Console window now also monitors the EJB server.

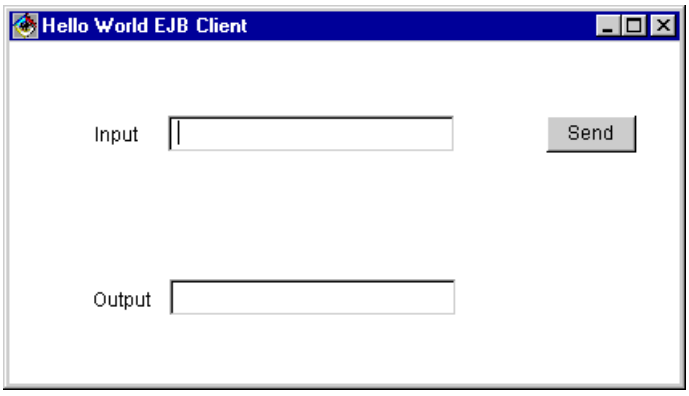
   (If you have problems starting the EJB server, in the Console window, select the EJB server process and examine the first error message that appears in the Output pane. If the error message indicates that the maximum number of database connections or applications has been exceeded, refer to the topic “Handling Problems Caused by DB2 Connection or Application Limits” for information on how to correct this problem. If, however, an error message indicates that the JDBC driver cannot be loaded, or if the debugger has opened, you should try creating a new project and importing the DB2 JDBC driver into the new project. In some situations, importing the JDBC driver may be necessary when working with entity enterprise beans. Note that once you have addressed the problem that caused the EJB server to fail to start, you should stop the Persistent Name Server, start it again, and then start the EJB server.)
7. In the Console, check that the EJB Server is ready to process a call from the client by selecting the EJB Server entry at the bottom of the All Programs pane and then looking for the following message in the Output pane:

   ```
   Server open for business...
   ```

(The message should appear in the last line at the bottom of the output text. However, it may take a moment or two to appear.)

Set up and run the Hello World client application:

1. Switch to the Visual Composition Editor. From the **Bean** menu, select **Run** > **Check Class Path**. (If a dialog appears prompting you to save the changes, click **Yes** to generate the run-time code.) The client Properties dialog appears.
2. Click the Class Path tab.
3. Beside the **Project Path** field, click the **Edit** button. The Class Path dialog appears.
4. Ensure that the **IBM WebSphere Test Environment** project, the **IBM EJB Tools** project, and the **IBM Enterprise Extension Libraries** project are selected, then click **OK**.
5. At the bottom of the dialog, ensure that the **Save in repository (as default)** check box is selected, then click **OK**.
6. On the Composition Editor tool bar, click the Run button. The Hello World EJB Client appears:



7. At the bottom of the All Programs pane in the Console window, select the EJB server process.
8. Position the Console window to the right of the screen.
9. Select the Hello World EJB Client window and position it to the left of the screen.
10. In the Hello World EJB Client, in the **Input** field, type `HelloWorld` and click the **Send** button.
11. In the **Output** field, the following string appears:

```
You said: HelloWorld
```

12. Close the Hello World EJB Client.

Stop the EJB server and remove the EJB server configuration:

1. In the EJB Server Configuration browser, select the EJB server (server1) in the Servers pane, click mouse button 2 and select **Stop Server** from the pop-up menu.
2. Select EJB Server (server1) again, click mouse button 2 and select **Remove Server**.

Conclusion

In this tutorial, you used the EJB Development Environment to develop a client application using an access bean and a session enterprise bean.

RELATED TASKS

Preparing for the tutorials
Tutorial: Increment
Handling problems caused by DB2 connection or application limits

Appendix. Reference Information

Type mappings for top-down programming

In Java, the `float` data type is not the same as the database FLOAT data type. The database equivalent of the Java `float` type is the REAL type. Note that the database types FLOAT and DOUBLE are synonyms and are the equivalent of the Java type `double`.

Java type	SQL number and type	Converter
java.lang.String	12 - VARCHAR(30)	
short or java.lang.Short	5 - SMALLINT	
int or java.lang.Integer	4 - INTEGER	
float or java.lang.Float	7 - REAL	
double or java.lang.Double	8 - DOUBLE	
byte or java.lang.Byte	5 - SMALLINT	VapNumberToByteConverter
char or java.lang.Character	1 - CHAR(1)	VapStringToCharacterConverter
boolean or java.lang.Boolean	5 - SMALLINT	VapNumberToBooleanConverter
java.math.BigDecimal	3 - DECIMAL(20, 2)	
java.sql.Date	91 - DATE	
java.sql.Time	92 - TIME	
java.sql.Timestamp	93 - TIMESTAMP	
java.util.Calendar	91 - DATE	VapDateToCalendarConverter
boolean[]	2004 - BLOB(2000)	VapBinaryStreamToSerializableObjectConverter
short[]	2004 - BLOB(2000)	VapBinaryStreamToSerializableObjectConverter
long[]	2004 - BLOB(2000)	VapBinaryStreamToSerializableObjectConverter
int[]	2004 - BLOB(2000)	VapBinaryStreamToSerializableObjectConverter
float[]	2004 - BLOB(2000)	VapBinaryStreamToSerializableObjectConverter
double[]	2004 - BLOB(2000)	VapBinaryStreamToSerializableObjectConverter
byte[]	2004 - BLOB(2000)	VapBinaryStreamToSerializableObjectConverter
char[]	2004 - BLOB(2000)	VapBinaryStreamToSerializableObjectConverter
java.io.Serializable	2004 - BLOB(2000)	VapBinaryStreamToSerializableObjectConverter

RELATED CONCEPTS

Approaches for mapping enterprise beans to database tables

RELATED TASKS

Generating a schema and mapping from an EJB group

RELATED REFERENCES

Type mappings for bottom-up programming
Default type mappings for deployment to Component Broker
Data-type converters supported for persistence

© Copyright IBM Corp. 1998, 2000

Type mappings for bottom-up programming

In Java, the `float` data type is not the same as the database FLOAT data type. The database equivalent of the Java `float` type is the REAL type. Note that the database types FLOAT and DOUBLE are synonyms and are the equivalent of the Java type `double`.

SQL number and type	Java type	Default converter
-5 - BIGINT	long	
-2 - BINARY	byte[]	
-7 - BIT	boolean	
2004 - BLOB	byte[]	
1 - CHAR	java.lang.String	VapTrimStringConverter
2005 - CLOB	java.lang.String	VapStringVarChar
91 - DATE	java.sql.Date	
3 - DECIMAL	java.math.BigDecimal	
8 - DOUBLE	double	
6 - FLOAT	double	
4 - INTEGER	int	
-4 - LONG VARBINARY	byte[]	
-1 - LONG VARCHAR	java.lang.String	
-95 - NCHAR	java.lang.String	
-97 - NTEXT	java.lang.String	
2 - NUMERIC	java.math.BigDecimal	
-96 - NVARCHAR	java.lang.String	
7 - REAL	float	
5 - SMALLINT	short	
92 - TIME	java.sql.Time	
93 - TIMESTAMP	java.sql.Timestamp	
-6 - TINYINT	byte	
-3 - VARBINARY	byte[]	
12 - VARCHAR	java.lang.String	

RELATED CONCEPTS

Approaches for mapping enterprise beans to database tables

RELATED TASKS

Creating EJB groups from schemas or models

RELATED REFERENCES

Type mappings for top-down programming
Default type mappings for deployment to Component Broker
Data-type converters supported for persistence

Default type mappings for deployment to Component Broker

If you wish to deploy your CMP beans to Component Broker, be aware that VisualAge for Java converters other than VapTrimStringConverter are not directly supported there. However, the Object Builder tool can automatically handle the mapping of most basic Java data types in preparation for deployment to Component Broker. The safest course of action is to check the schema created in Object Builder.

If the data types used in your CMP beans are not included in the following table, you can still deploy them to Component Broker, but you must map the types yourself in Object Builder. Otherwise, user-defined types are interpreted as Serializable and stored as a binary VARBINARY field in the database.

Additional SQL types may be supported by Object Builder for use with DB2 databases. For more information, see the Object Builder documentation.

In Java, the `float` data type is not the same as the database FLOAT data type. The database equivalent of the Java `float` type is the REAL type. Note that the database types FLOAT and DOUBLE are synonyms and are the equivalent of the Java type `double`.

Java type	**JDBC type**
java.lang.String	VARCHAR (VARGRAPHIC)
short	SMALLINT
int	INTEGER
long	INTEGER
float	REAL
double	DOUBLE
byte	SMALLINT
char	CHAR(1)
boolean	SMALLINT
java.math.BigDecimal	DECIMAL
java.math.BigInteger	DECIMAL
java.sql.Date	DATE
java.sql.Time	TIME
java.sql.Timestamp	TIMESTAMP
byte[]	VARBINARY

RELATED CONCEPTS

Approaches for mapping enterprise beans to database tables

RELATED TASKS

Generating a schema and mapping from an EJB group

RELATED REFERENCES

Type mappings for bottom-up programming
Type mappings for top-down programming
Data-type converters supported for persistence

Type compatibility by database vendor

This topic provides information on data-type support for persistence in the EJB Development Environment. When defining schemas, be sure to consider compatibility issues as well as how the types will be used within keys. For optimal performance, use integers for keys wherever possible, because they are hashed quickly and database lookup is much faster than with other types.

Problems with using certain types as key values

The VisualAge for Java development tools enable you to select any SQL data type to use as a primary key for a table as well as any object type to use in the key of a persistent class or enterprise bean. Be aware of some problems that may occur when using particular types in keys.

Some databases do not allow for BLOB types to be declared within the primary key of the table; however, binary types may be allowed. Consult your database vendor documentation to determine what SQL data types are allowed.

In certain situations, the performance of your entity beans might be significantly affected by the hash performance of their keys. Ideally, the key's hashValue() method will be fast and produce a smooth distribution of hash values. Of course, the hashValue() method must produce the same value for any two instances for which the equals() method returns true. The key's hashValue() method is one place to check for performance problems. In particular, using arrays as key fields sometimes causes problems. The default hashValue() and equals() methods in the Java language use the array's address as the hash value and for comparisons. These methods do not examine the contents of the array. For a key, this is probably not the desired behavior. You might prefer to wrap the array with another object, write your own hashValue() and equals() methods in the wrapper, and use this wrapper type as the bean's field type. For CMP entity beans, the generated hashValue() and equals() methods in the key class would then use these methods. For CMP entity beans, you might need a converter for the corresponding database column. Finally, if you do write a hashValue() method that examines the elements of an array, you might want to consider not examining all elements of large arrays.

Another type to avoid using in the key of a persistent class or enterprise bean is a numeric type with scale, such as java.math.BigDecimal. This type is problem-prone because of hashing behavior as well. The hashing problem stems from the fact that numbers of different scale get hashed differently even when they have the same numeric value (for example, 12.0 versus 12.000). This causes run-time problems, because the scale that is returned from the database may be different from the scale that was inserted.

Type compatibility table

In the following table, support of the specified type is indicated by a filled circle (●).

Note: The underlying databases might support more types than what is indicated. The respective JDBC drivers limit the use of all types in some cases.

JDBC type (number)	DB2/UDB	DB2/390	DB2/400	Oracle	Sybase
BIT (-7)				●	●
TINYINT (-6)				●	●
SMALLINT (5)	●	●	●	●	●
INTEGER (4)	●	●	●	●	●
BIGINT (-5)	●			●	
FLOAT (6)	●	●	●	●	●
REAL (7)	●	●	●	●	●
DOUBLE (8)	●	●	●		●
NUMERIC (2)	●	●	●	●	●
DECIMAL (3)	●	●	●		●
CHAR (1)	●	●	●	●	●
VARCHAR (12)	●	●	●	●	●
LONGVARCHAR (-1)	●	●		●	●
DATE (91)	●	●	●		
TIME (92)	●	●	●		
TIMESTAMP (93)	●	●	●	●	●
BINARY (-2)	●	●	●		●
VARBINARY (-3)	●	●	●	●	●
LONGVARBINARY (-4)	●	●		●	●
JDBC 2.0 types					
ARRAY (2003)					
BLOB (2004)	●	●	●	●	
CLOB (2005)	●	●	●	●	
REF (2006)					
Unicode/double-byte types					
NCHAR(1)	●				●
NVARCHAR(12)	●				●
NLONGVARCHAR	●				●

DDL type compatibility table

In the following table, **bold face** in an entry indicates type substitution. In the other cases, the types are compatible by default.

JDBC type	DB2/UDB	DB2/390	DB2/400	Oracle	Sybase
BIT (-7)	**SMALLINT**	**SMALLINT**	**SMALLINT**	NUMBER (1)	bit
TINYINT (-6)	**SMALLINT**	**SMALLINT**	**SMALLINT**	NUMBER (3)	tinyint

JDBC type	DB2/UDB	DB2/390	DB2/400	Oracle	Sybase
SMALLINT (5)	SMALLINT	SMALLINT	SMALLINT	NUMBER (5)	smallint
INTEGER (4)	INTeger	INTeger	INTeger	NUMBER (10)	int
BIGINT (-5)	BIGINT	**NUMERIC (38)**	**NUMERIC (38)**	NUMBER (38)	**numeric (38)**
FLOAT (6)	FLOAT	FLOAT (15)	FLOAT	FLOAT	float
REAL (7)	REAL	REAL	REAL	REAL	real
DOUBLE (8)	DOUBLE	DOUBLE PRECISION	DOUBLE	**FLOAT**	double precision
NUMERIC (2)	NUMERIC (prec,scale)	NUMERIC (prec,scale)	NUMERIC (prec,scale)	NUMBER (prec,scale)	numeric (prec,scale)
DECIMAL (3)	DECIMAL (prec,scale)	DECIMAL (prec,scale)	DECIMAL (prec,scale)	**NUMBER (prec,scale)**	decimal (prec,scale)
CHAR (1)	CHAR (length)	CHAR (length)	CHAR (length)	CHAR (length)	char (length)
VARCHAR (12)	VARCHAR (length)	VARCHAR (length)	VARCHAR (length)	VARCHAR2 (length)	varchar (length)
LONGVARCHAR (-1)	LONG VARCHAR	LONG VARCHAR	LONG VARCHAR	LONG	text
DATE (91)	DATE	DATE	DATE	**DATE**	**datetime**
TIME (92)	TIME	TIME	TIME	**DATE**	**datetime**
TIMESTAMP (93)	TIMESTAMP	TIMESTAMP	TIMESTAMP	DATE	datetime
BINARY (-2)	CHAR (length) FOR BIT DATA	CHAR (length) FOR BIT DATA	CHAR (length) FOR BIT DATA	**RAW (length)**	binary (length)
VARBINARY (-3)	VARCHAR (length) FOR BIT DATA	VARCHAR (length) FOR BIT DATA	VARCHAR (length) FOR BIT DATA	RAW (length)	varbinary (length)
LONGVARBINARY (-4)	LONG VARCHAR FOR BIT DATA	LONG VARCHAR FOR BIT DATA	**LONG VARCHAR FOR BIT DATA**	LONG RAW	image
JDBC 2.0 types					
ARRAY (2003)					
BLOB (2004)	BLOB (length)	BLOB (length)	BLOB (length)	**BLOB**	image
CLOB (2005)	CLOB (length)	CLOB (length)	CLOB (length)	**CLOB**	text
REF (2006)					
Unicode/double-byte types (no JDBC number)					
NCHAR	GRAPHIC (length)				nchar (length)
NVARCHAR	VARGRAPHIC (length)				nvarchar (length)

JDBC type	DB2/UDB	DB2/390	DB2/400	Oracle	Sybase
NLONGVARCHAR	LONGVARGRAPHIC				ntext

Finder helpers generated for deployment to Component Broker

When you export code for use in the Component Broker tools of the WebSphere Application Server, VisualAge for Java generates finder-helper implementation code for the following items:

Associations. VisualAge for Java generates implementation code for the finder methods on the home interface that correspond to associations every time you export the enterprise bean. Do not attempt to write these yourself.

Custom finders on the home interface. VisualAge for Java generates custom finder methods in a finder helper class for each custom finder method on the home interface. The generated finder helper classes specific to Component Broker are loaded into the Workbench after exportation to Component Broker. You can browse them on the EJB page when you select the option to show generated classes.

If VisualAge for Java detects a corresponding Component Broker query field in the finder helper interface, it uses that field to construct a working Component Broker finder method. If VisualAge for Java detects a custom finder method on the home interface without a corresponding Component Broker query field on the finder helper interface, it generates a stub method. It is the responsibility of the user to implement the Component Broker finder method. While the stub method exists in the finder helper class, it is not regenerated. If you change the implementation of any of these stub methods, you will need to export the EJB group to a JAR file for Component Broker.

Both Component-Broker and non-Component-Broker finder query fields can coexist on the finder helper interface.

VisualAge for Java puts the newly generated method code into the package in which the enterprise bean's bean class resides. If no open edition exists at generation time, VisualAge for Java opens a new edition.

Four different Component Broker query types are supported on the finder helper interface in VisualAge for Java. The value of the query field must be a valid OO-SQL expression. Usage of a given query type causes VisualAge for Java to generate calls to the Component Broker evaluation method listed in the following table. *xxx* indicates the rest of the query-field name.

Query-field name	Component Broker call generated
*xxx*CBWhereClause	• evaluate(String) if an Enumeration is returned • singleEvaluate(String) if a single object is returned
*xxx*CBQueryString	• extendedEvaluate(String) if an Enumeration is returned • extendedSingleEvaluate(String) if a single object is returned
*xxx*CBLazyWhereClause	lazyEvaluate(String)
*xxx*CBLazyQueryString	extendedLazyEvaluate(String)

For example, if you have a custom finder method on the home interface named findCustomersByName, you could have the corresponding CB query field named findCustomersByNameCBWhereClause on the finder helper interface.

For more information about Component Broker queries, Component Broker custom finder methods, and evaluation methods, see the documentation for the WebSphere Application Server, Enterprise Edition.

RELATED TASKS

Exporting enterprise beans to EJB or deployed JAR files
Creating or editing associations

Example: Wrapping a stored procedure in a session bean

This topic illustrates the use of a DB2 stored procedure with a session-bean wrapper. Two versions of the example are provided. One version wraps a stored procedure using JDBC only; this is the simplest case. The other version uses a servlet and JNDI to handle payroll-calculation requests.

The stored procedure, DEPTPAYROLL, retrieves salaries for all employees in a department, adds them up, and returns the sum.

For more information on building stored procedures, read the tutorial "Building Stored Procedures in VisualAge for Java 3.0" on IBM's VisualAge Developer Domain site (http://www.ibm.com/vadd).

First, take a look at the stored procedure itself:

```
package myProcedures;
import java.sql.*;                    // JDBC classes
public class DEPTPAYROLL{
   public static void dEPTPAYROLL ( String DEPTNO,
                                    String[] PAYROLL )
                      throws SQLException, Exception{

      // get connection to the database
      Connection con = DriverManager.getConnection
                   ("jdbc:default:connection");
      PreparedStatement stmt = null;
      ResultSet rs = null;
      String sql;

      // add up individual salaries for WORKDEPT
      sql = "SELECT"
          + " SUM(EMPLOYEE.SALARY)"
          + " FROM"
          + " EMPLOYEE"
          + " WHERE"
          + " ("
          + " ( "
          + " EMPLOYEE.WORKDEPT = ? "
          + " )"
          + " )";

      stmt = con.prepareStatement( sql );
      stmt.setString( 1, DEPTNO );
      rs = stmt.executeQuery();

      // access query results (only 1 element in array)
      while (rs.next()) {
      PAYROLL[0] = rs.getString(1);
      }
```

```
      // close open resources
      if (rs != null) rs.close();
      if (stmt != null) stmt.close();
      if (con != null) con.close();

   }
}
```

DEPTPAYROLL does the following:

1. Opens a default connection.
2. Builds the SQL query string template.
3. Sets the only input parameter to the SQL query string by passing in its first input parameter, DEPTNO, a String department ID.
4. Executes the query.
5. Returns the result set. Since the sum is done as part of the query, the result set includes exactly one row.

From here, refer to the following topics:

"Wrapping a stored procedure using JDBC only" describes the simplest case. The session bean opens a connection through JDBC and calls the stored procedure.

"Wrapping a stored procedure using J2EE" uses a servlet to handle payroll-calculation requests. The servlet locates the session bean using JNDI lookup, creates an instance, and calls the session bean's calculation method. In turn, the session bean opens a connection using a static DataSource object and calls the stored procedure.

RELATED REFERENCES

Wrapping a stored procedure using JDBC only
Wrapping a stored procedure using J2EE

Wrapping a stored procedure using JDBC only

This topic illustrates a session-bean wrapper that uses JDBC only. The session bean (DepartmentUtil) opens a JDBC connection and calls a stored procedure. The stored procedure, DEPTPAYROLL, retrieves salaries for all employees in a department, adds them up, and returns the sum.

Session bean implementation

The session bean's bean class (DepartmentUtilBean) contains a method, calculatePayroll(String), that calls the stored procedure. The calculatePayroll(String) method does the following:

1. Opens a connection to a DB2 database using JDBC.
2. Builds a string template for the stored-procedure call.
3. Sets the only input parameter to the stored-procedure call string by passing in its only input parameter, DepartmentId (DEPTNO within the stored procedure).
4. Registers the type of the output, a VARCHAR.
5. Calls the stored procedure.
6. Captures the out parameter, known within the stored procedure as PAYROLL.

```
public String calculatePayroll(String DepartmentId)
                    throws java.rmi.RemoteException {

   String payroll = null; // variable to hold the result
```

```
   try {
      String url = "jdbc:db2:sample";
      Class.forName("COM.ibm.db2.jdbc.app.DB2Driver").newInstance();
      Connection con = DriverManager.getConnection(url);  // get default connection

      // assemble String that defines the procedure call
      String sql = "Call " + "DEPTPAYROLL" + "(?,?) ";

      // build a callable statement
      CallableStatement stmt = con.prepareCall (sql);
      stmt.setString (1, DepartmentId); // set the Department ID
      stmt.registerOutParameter(2,Types.VARCHAR); // set up "out" parameter

      // call the stored procedure
      stmt.execute ();

      // get the second "out" parameter returned by the stored procedure
      payroll = stmt.getString(2);

      // close
      stmt.close ();
      con.close ();

   }
   catch( Exception e ) {
      e.printStackTrace();
   }

   // return the sum
   return payroll;
}
```

RELATED REFERENCES

Wrapping a stored procedure in a session bean
Wrapping a stored procedure using J2EE

Wrapping a stored procedure using J2EE

This topic illustrates a session-bean wrapper that uses J2EE with JDBC in a WebSphere Application Server, Version 3.5, environment. A servlet (PayrollServlet2) handles payroll-calculation requests. The session bean (DepartmentUtil2) establishes a JDBC connection using a DataSource object and calls the stored procedure. The stored procedure, DEPTPAYROLL, retrieves salaries for all employees in a department, adds them up, and returns the sum. The following discussion begins at the stored procedure and traces back through the session bean and servlet classes.

Session bean implementation

All code relevant to this discussion resides in the bean class, DepartmentUtil2Bean. First, note the packages being imported; this implementation will not work in versions of WebSphere Application Server prior to 3.5:

```
import java.util.*;
import javax.naming.*;

// J2EE Standard packages for WebSphere 3.5
import javax.sql.*;
import javax.transaction.*;
```

The DataSource object, ds, is declared in the bean class as follows:

```
private static DataSource ds = null;
```

The session bean's creation method sets up the context for JNDI lookup and connection to the data source, as follows:

```
public void ejbCreate() throws javax.ejb.CreateException,
                               java.rmi.RemoteException {
   if (ds == null) {
      try {
         System.out.println("Creating Initial Context...");
         Context ctx = new InitialContext();
         System.out.println("initial context created");

         //Look up our datasource (Type/DataSourceName)
         ds = (DataSource) ctx.lookup("jdbc/sample");
         if (ds == null) System.out.println("No data source found");

         System.out.println("Driver Manager is Using DataSource:" +
                            ds.toString() );
      }
      catch (Exception e) {
         System.out.println("Naming service exception at ejbCreate(): " +
                            e.getMessage() );
      }
   }
}
```

The calculatePayroll(String) method connects to the initialized data source and calls the stored procedure, as follows:

1. Opens a connection to a database through the DataSource object created and initialized in the ejbCreate() method.
2. Builds a string template for the stored-procedure call.
3. Sets the only input parameter to the stored-procedure call string by passing in its only input parameter, DepartmentId (DEPTNO within the stored procedure).
4. Registers the type of the output, a VARCHAR.
5. Calls the stored procedure.
6. Captures the out parameter, known within the stored procedure as PAYROLL.

```
public String calculatePayroll(String DepartmentId)
                    throws java.rmi.RemoteException {

   String payroll = null; // variable to hold the result

   try {
      System.out.println("calculatePayroll, get connection from WebSphere data source");
      java.sql.Connection con = ds.getConnection();

      // assemble String that defines the procedure call
      String sql = "Call " + "DEPTPAYROLL" + "(?,?) ";

      // build a callable statement
      CallableStatement stmt = con.prepareCall (sql);
      stmt.setString (1, DepartmentId); // set the Department ID
      stmt.registerOutParameter(2,Types.VARCHAR); // set up "out" parameter

      // call the stored procedure
      stmt.execute ();

      // get the second "out" parameter returned by the stored procedure
      payroll = stmt.getString(2);

      // close
      stmt.close ();
      con.close ();

   }
```

```
   catch( Exception e ) {
      System.out.println("Exception in Calculate Payroll");
      e.printStackTrace();
   }

   // return the sum
   return payroll;
}
```

Servlet implementation

An extension of the HttpServlet class, PayrollServlet2 does its relevant work in the doGet method, as follows:

1. Retrieves the department ID parameter from the HTTP request as departmentNumber.
2. Locates the session bean and creates an instance.
3. Calls the session bean's calculatePayroll(String) method, passing in departmentNumber.

```
public void doGet(HttpServletRequest req,
                  HttpServletResponse res) throws IOException {

   // Set up the output stream
   PrintWriter out = res.getWriter();
   res.setContentType("text/plain");

   // Get incoming parameter from the Request Object
   String departmentNumber = req.getParameter("departmentNumber");
   DepartmentUtil2Home home = null;

   try {
      InitialContext ctx = null;
      out.println("Creating Initial Context...");
      ctx = new InitialContext();

      out.println("Creating HomeObject...");
      Object homeObject = ctx.lookup("DepartmentUtil2");

      out.println("Narrow HomeObject to DepartmentUtil2Home...");
      home = (DepartmentUtil2Home)javax.rmi.PortableRemoteObject.narrow(
              homeObject, DepartmentUtil2Home.class);

      out.println("Creating the EJB DepatmentUtil2 ...");
      DepartmentUtil2 dept = home.create();

      // call domain method to do some real work
      out.println("Calculating the payroll for department: "+ departmentNumber);
      String payroll = dept.calculatePayroll(departmentNumber);

      if (payroll != null) {
         out.println("The payroll for department: "+ departmentNumber +
                                 " is : " + payroll);
      } else {
         out.println("Department: "+ departmentNumber +
                     " is not a valid department ");
      }

      // destroy ejb
      out.println("Clean up, removing ejb ");
      dept.remove();
      out.println("Done.. " );
      out.close();

   }
   catch (javax.naming.NamingException e) {
      out.println("DepartmentUtil2 naming exception");
```

```
            e.printStackTrace();

        }
        catch (RemoteException e) {
            out.println("Remote Exception");
            e.printStackTrace();

        }
        catch (NullPointerException e) {
            if (home == null)
                out.println( "Cannot find EJB's home. " + e.getMessage() );
            else
                e.printStackTrace();
        }
        catch (Exception e) {
            out.println( "General Exception " + e.getMessage() );
            e.printStackTrace();
        }
    }
```

RELATED REFERENCES

Example: Wrapping a stored procedure in a session bean
Wrapping a stored procedure using JDBC only

Example: Using copy helper and rowset access beans

This example shows you how to use copy helper and rowset access beans in a simple business scenario. To fully understand the example, you should already have some familiarity with the EJB Development Environment conceptual information on EJB technology, inheritance, associations, and access beans.

For the purposes of the example, assume that you have three CMP entity beans with the following names:

- Employee
- Manager
- Department

All of the classes reside in a package named empexample.

The CMP fields and relationships for these three CMP beans are shown in the following table, as well as the type of access bean generated for each of the CMP beans:

	Employee	**Manager**	**Department**
CMP Fields	id name salary	parkinglotnum	id name projectcode
Relationships	Superclass of Manager Participates in 1:N association with Department. (Employee has one department, but Department has many employees.)	Subclass of Employee (inherits from employee)	Participates in 1:N association with Employee. (Department has many employees, but Employee has one department.)
Access bean	Rowset	Copy helper	Copy helper

Example code relating to these three CMP beans is provided in the following sections:

- EJB interfaces (page 160)
- Access bean method signatures (page 161)
- Client program (page 164)

EJB interfaces

In the following code example, the interfaces for the three CMP entity beans are provided:

```
public interface Employee extends javax.ejb.EJBObject {
  empexample.Department getDepartment() throws java.rmi.RemoteException,
     javax.ejb.FinderException;
  empexample.DepartmentKey getDepartmentKey() throws java.rmi.RemoteException;
  java.lang.String getName() throws java.rmi.RemoteException;
  float getSalary() throws java.rmi.RemoteException;
  void privateSetDepartmentKey(empexample.DepartmentKey inKey) throws
     java.rmi.RemoteException;
  void secondarySetDepartment(empexample.Department aDepartment) throws
     java.rmi.RemoteException;
  void setDepartment(empexample.Department aDepartment) throws
     java.rmi.RemoteException;
  void setName(java.lang.String newValue) throws java.rmi.RemoteException;
  void setSalary(float newValue) throws java.rmi.RemoteException;
}

public interface EmployeeHome extends javax.ejb.EJBHome {
  empexample.Employee create(int argId) throws javax.ejb.CreateException,
     java.rmi.RemoteException;
  empexample.Employee create(int argId, int depId) throws
     javax.ejb.CreateException, java.rmi.RemoteException;
  empexample.Employee findByPrimaryKey(empexample.EmployeeKey key) throws
     java.rmi.RemoteException, javax.ejb.FinderException;
  java.util.Enumeration findEmployeeByDepartment(empexample.DepartmentKey inKey)
     throws java.rmi.RemoteException, javax.ejb.FinderException;
}

public interface Manager extends Employee {
  int getParkinglotnum() throws java.rmi.RemoteException;
  void setParkinglotnum(int newValue) throws java.rmi.RemoteException;
}

public interface ManagerHome extends javax.ejb.EJBHome {
  empexample.Manager create(int argId) throws javax.ejb.CreateException,
     java.rmi.RemoteException;
  empexample.Manager findByPrimaryKey(EmployeeKey key) throws
     java.rmi.RemoteException, javax.ejb.FinderException;
}

public interface Department extends javax.ejb.EJBObject {
  void addEmployee(empexample.Employee anEmployee) throws
     java.rmi.RemoteException;
  java.util.Enumeration getEmployee() throws java.rmi.RemoteException,
     javax.ejb.FinderException;
  java.lang.String getName() throws java.rmi.RemoteException;
  int getProjectcode() throws java.rmi.RemoteException;
  void secondaryAddEmployee(empexample.Employee anEmployee) throws
     java.rmi.RemoteException;
  void secondaryRemoveEmployee(empexample.Employee anEmployee) throws
     java.rmi.RemoteException;
  void setName(java.lang.String newValue) throws java.rmi.RemoteException;
  void setProjectcode(int newValue) throws java.rmi.RemoteException;
}
```

```
public interface DepartmentHome extends javax.ejb.EJBHome {
  empexample.Department create(int argId) throws javax.ejb.CreateException,
     java.rmi.RemoteException;
  empexample.Department findByPrimaryKey(empexample.DepartmentKey key) throws
     java.rmi.RemoteException, javax.ejb.FinderException;
}
```

Access bean method signatures

At this point, assume your three CMP entity beans are complete. The Manager CMP entity bean inherits from the Employee CMP entity bean, and the Employee CMP entity bean has a 1:N association with the Department CMP entity bean.

You would then generate your access beans: a copy helper access bean for the Department and Manager CMP beans, and a rowset access bean for the Employee CMP bean. You can assume that all CMP fields (except the getters for the key of the associated CMP entity beans) are cached without using string converters for the getters and setters.

In the following code example, the method signatures for the access beans are provided:

```
public class EmployeeAccessBean extends com.ibm.ivj.ejb.runtime.AbstractEntityAccessBean
    implements EmployeeAccessBeanData {

  /**
   * Zero argument constructor used to initialize the access bean.
   *
   * This constructor corresponds to the following home interface method:
   *
   * public abstract empexample.Employee empexample.EmployeeHome.create(int)
   *     throws javax.ejb.CreateException,java.rmi.RemoteException
   *
   * The home interface method properties need to be set by calling
   * the following setter methods before calling any business methods:
   * setInit_argId( int )
   * setInit_depId( int )
   */
   public EmployeeAccessBean()
   public EmployeeAccessBean(empexample.EmployeeKey arg0) throws
      java.rmi.RemoteException, javax.ejb.FinderException,
      javax.naming.NamingException
   public EmployeeAccessBean ( javax.ejb.EJBObject o ) throws
      java.rmi.RemoteException
   public EmployeeAccessBean ( int arg0 ) throws
      javax.ejb.CreateException, java.rmi.RemoteException,
      javax.naming.NamingException
   public void commitCopyHelper() throws java.rmi.RemoteException,
      javax.ejb.CreateException, javax.ejb.FinderException,
      javax.naming.NamingException
   protected String defaultJNDIName()
   private empexample.EmployeeHome ejbHome() throws
      private.rmi.RemoteException, javax.naming.NamingException
   private empexample.Employee ejbRef() throws java.rmi.RemoteException
   public java.util.Enumeration findEmployeeByDepartment(empexample.DepartmentKey arg0)
      throws java.rmi.RemoteException, javax.ejb.FinderException,
      javax.naming.NamingException
   public empexample.DepartmentAccessBean getDepartment() throws
      java.rmi.RemoteException, javax.ejb.FinderException,
      javax.ejb.CreateException, javax.naming.NamingException
   public empexample.DepartmentKey getDepartmentKey() throws
      java.rmi.RemoteException, javax.ejb.CreateException,
      javax.naming.NamingException
   public java.lang.String getName() throws java.rmi.RemoteException,
      javax.ejb.CreateException, javax.ejb.FinderException,
```

```
      javax.naming.NamingException
   public float getSalary() throws java.rmi.RemoteException,
      javax.ejb.CreateException, javax.ejb.FinderException,
      javax.naming.NamingException
   protected void instantiateEJB() throws javax.ejb.CreateException,
      java.rmi.RemoteException, javax.naming.NamingException
   protected boolean instantiateEJBByPrimaryKey() throws
      java.rmi.RemoteException, javax.ejb.CreateException,
      javax.naming.NamingException
   public void privateSetDepartmentKey(empexample.DepartmentKey arg0)
      throws java.rmi.RemoteException, javax.ejb.CreateException,
      javax.naming.NamingException
   public void refreshCopyHelper() throws java.rmi.RemoteException,
      javax.ejb.CreateException, javax.ejb.FinderException,
      javax.naming.NamingException
   public void secondarySetDepartment(empexample.Department arg0)
      throws java.rmi.RemoteException, javax.ejb.CreateException,
      javax.naming.NamingException
   public void setDepartment(empexample.Department arg0) throws
      java.rmi.RemoteException, javax.ejb.CreateException,
      javax.naming.NamingException
   public void setInit_argId( int newValue )
   public void setInit_depId( int newValue )
   public void setName( java.lang.String newValue )
   public void setSalary( float newValue )
}
public class ManagerAccessBean extends
    com.ibm.ivj.ejb.runtime.AbstractEntityAccessBean implements ManagerAccessBeanData {
  /**
   * Zero argument constructor used to initialize the access bean.
   *
   * This constructor corresponds to the following home interface method:
   *
   * public abstract empexample.Manager empexample.ManagerHome.create(int)
   *    throws javax.ejb.CreateException,java.rmi.RemoteException
   *
   * The home interface method properties need to be set by calling
   * the following setter methods before calling any business methods:
   * setInit_argId( int )
   */
   public ManagerAccessBean ()
   public ManagerAccessBean(empexample.EmployeeKey arg0) throws
      java.rmi.RemoteException, javax.ejb.FinderException,
      javax.naming.NamingException
   public ManagerAccessBean ( javax.ejb.EJBObject o ) throws
      java.rmi.RemoteException
   public void commitCopyHelper() throws java.rmi.RemoteException,
      javax.ejb.CreateException, javax.ejb.FinderException,
      javax.naming.NamingException
   protected String defaultJNDIName()
   private empexample.ManagerHome ejbHome() throws
      java.rmi.RemoteException, javax.naming.NamingException
   private empexample.Manager ejbRef() throws java.rmi.RemoteException
   public empexample.DepartmentAccessBean getDepartment() throws
      java.rmi.RemoteException, javax.ejb.FinderException,
      javax.ejb.CreateException, javax.naming.NamingException
   public empexample.DepartmentKey getDepartmentKey() throws
      java.rmi.RemoteException, javax.ejb.CreateException,
      javax.naming.NamingException
   public java.lang.String getName() throws java.rmi.RemoteException,
      javax.ejb.CreateException, javax.ejb.FinderException,
      javax.naming.NamingException
   public int getParkinglotnum() throws java.rmi.RemoteException,
      javax.ejb.CreateException, javax.ejb.FinderException,
      javax.naming.NamingException
   public float getSalary() throws java.rmi.RemoteException,
```

```
      javax.ejb.CreateException, javax.ejb.FinderException,
      javax.naming.NamingException
   protected void instantiateEJB() throws javax.ejb.CreateException,
      java.rmi.RemoteException, javax.naming.NamingException
   protected boolean instantiateEJBByPrimaryKey() throws
      java.rmi.RemoteException, javax.ejb.CreateException,
      javax.naming.NamingException
   public void privateSetDepartmentKey(empexample.DepartmentKey arg0)
      throws java.rmi.RemoteException, javax.ejb.CreateException,
      javax.naming.NamingException
   public void refreshCopyHelper() throws java.rmi.RemoteException,
      javax.ejb.CreateException, javax.ejb.FinderException,
      javax.naming.NamingException
   public void secondarySetDepartment(empexample.Department arg0)
      throws java.rmi.RemoteException, javax.ejb.CreateException,
      javax.naming.NamingException
   public void setDepartment(empexample.Department arg0) throws
      java.rmi.RemoteException, javax.ejb.CreateException,
      javax.naming.NamingException
   public void setInit_argId( int newValue )
   public void setName( java.lang.String newValue )
   public void setParkinglotnum( int newValue )
   public void setSalary( float newValue )
}
public class DepartmentAccessBean extends
   com.ibm.ivj.ejb.runtime.AbstractEntityAccessBean implements DepartmentAccessBeanData {
  /**
   * Zero argument constructor used to initialize the access bean.
   *
   * This constructor corresponds to the following home interface method:
   *
   * public abstract empexample.Department empexample.DepartmentHome.create(int)
   *    throws javax.ejb.CreateException,java.rmi.RemoteException
   *
   * The home interface method properties need to be set by calling
   * the following setter methods before calling any business methods:
   * setInit_argId( int )
   */
   public DepartmentAccessBean ()
   public DepartmentAccessBean(empexample.DepartmentKey arg0) throws
      java.rmi.RemoteException, javax.ejb.FinderException,
      javax.naming.NamingException
   public DepartmentAccessBean ( javax.ejb.EJBObject o ) throws
      java.rmi.RemoteException
   public void addEmployee(empexample.Employee arg0) throws
      java.rmi.RemoteException, javax.ejb.CreateException,
      javax.naming.NamingException
   public void commitCopyHelper() throws java.rmi.RemoteException,
      javax.ejb.CreateException, javax.ejb.FinderException,
      javax.naming.NamingException
   protected String defaultJNDIName()
   private empexample.DepartmentHome ejbHome() throws
      java.rmi.RemoteException, javax.naming.NamingException
   private empexample.Department ejbRef() throws
      java.rmi.RemoteException
   public java.util.Enumeration getEmployee() throws
      java.rmi.RemoteException, javax.ejb.FinderException,
      javax.ejb.CreateException, javax.naming.NamingException
   public java.lang.String getName() throws java.rmi.RemoteException,
      javax.ejb.CreateException, javax.ejb.FinderException,
      javax.naming.NamingException
   public int getProjectcode() throws java.rmi.RemoteException,
      javax.ejb.CreateException, javax.ejb.FinderException,
      javax.naming.NamingException
   protected void instantiateEJB() throws javax.ejb.CreateException,\
      java.rmi.RemoteException, javax.naming.NamingException
```

```
   protected boolean instantiateEJBByPrimaryKey() throws
      java.rmi.RemoteException, javax.ejb.CreateException,
      javax.naming.NamingException
   public void refreshCopyHelper() throws java.rmi.RemoteException,
      javax.ejb.CreateException, javax.ejb.FinderException,
      javax.naming.NamingException
   public void secondaryAddEmployee(empexample.Employee arg0) throws
      java.rmi.RemoteException, javax.ejb.CreateException,
      javax.naming.NamingException
   public void secondaryRemoveEmployee(empexample.Employee arg0) throws
      java.rmi.RemoteException, javax.ejb.CreateException,
      javax.naming.NamingException
   public void setInit_argId( int newValue )
   public void setName( java.lang.String newValue )
   public void setProjectcode( int newValue )
}
```

Client program

In the following code example, a client program is shown performing various operations on the access beans:

```
package empexample;

import javax.ejb.*;
import com.ibm.ivj.ejb.runtime.*;

public class EmpDepTest {

/**
 * Simple test of the Employee, Department, and Manager
 * access beans.
 *
 * @param args an array of command-line arguments
 * args[0] = Employee ID#
 * args[1] = Employee Name
 * args[2] = Employee Salary
 * args[3] = Department ID#
 * args[4] = Parking lot# (optional)
 */

public static void main(java.lang.String[] args) {

    EmployeeAccessBean empab = null;
    ManagerAccessBean mgrab = null;
    DepartmentAccessBean depab = null;
    int empid;
    String empname;
    float empsalary;
    int mgrparknum;
    int depid;

    try {
        empid = Integer.parseInt(args[0]);
        empname = args[1];
        empsalary = Float.parseFloat(args[2]);
        depid = Integer.parseInt(args[3]);
        mgrparknum = Integer.parseInt(args[4]);

        // Attempt to create a new employee with given info
        try {
            empab = new EmployeeAccessBean();
            empab.setInit_argId(empid);
            empab.setInit_depId(depid);
```

```
    // Set the various attributes
                                                    // (gets written to cache)
    empab.setName(empname);
    empab.setSalary(empsalary);

    // Flush the cache to the server
    empab.commitCopyHelper();
}

// If duplicate key exception occurs, find the
// pre-existing instance instead
catch ( DuplicateKeyException dke ) {
        empab = new EmployeeAccessBean(new EmployeeKey(empid));

        // Fill the cache with all the attributes
        empab.refreshCopyHelper();

        // Update the pre-existing Employee with new info
        // and update the entity bean by flushing the cache
        empab.setName(empname);
        empab.setSalary(empsalary);
        empab.commitCopyHelper();
}

// Display employee info
// Get the Employee bean's key (assume key class has
// getters for key fields)
EmployeeKey empkey = (EmployeeKey) empab.__getKey();
System.out.println("Employee ID#: " + empkey.getId());
System.out.println("Employee Name: " + empab.getName());
System.out.println("Employee Salary: " + empab.getSalary());

// Get the Department access bean (for associated EJB) for
// this Employee
depab = empab.getDepartment();
if ( depab != null ) {
    DepartmentKey depkey = (DepartmentKey) depab.__getKey();

    // Find all the employees in this department
    // This is also shows the use of the new AccessBeanEnumeration
    // class which is a special enumeration class that only
    // instantiates the EJB object when nextElement() is called.
    // All access bean finder methods returning Enumerations of
    // EJBObject's now return this special enumeration class
    System.out.println("\nFinding all employees for department "
                                        + depab.getName());
    AccessBeanEnumeration aem =
        (AccessBeanEnumeration) empab.findEmployeeByDepartment(depkey);

    // Use an access bean table (rowset) to organize and manipulate the
    // enumeration of Employee access beans.
    // Usually, a session bean would first create this table before
    // passing it on to a JSP where the enumeration of access beans
    // can be handled like a rowset using indexes.
    // Rows (or EJB instances) can then be added or removed from the
    // rowset.
    EmployeeAccessBeanTable emptable = new EmployeeAccessBeanTable();

    // One possible way of filling the table is to call the method below
    //emptable.setEmployeeAccessBean(aem);

    // This is another way
    while ( aem.hasMoreElements() ) {
            EmployeeAccessBean empab_temp = (EmployeeAccessBean) aem.nextElement();
            emptable.addRow(empab_temp);
            empab_temp.refreshCopyHelper();
            System.out.println("    Employee Name: " + empab_temp.getName());
```

```
                    System.out.println("    Employee Salary: " + empab_temp.getSalary());
                }

                // Once the table is built, the client can go about working with
                // with it and performing various operations on it without any
                // server-side communications

            }

            else {
                 System.out.println("Could not find a department for employee id#" + empid);
            }

        }


        catch ( Exception e ) {
                e.printStackTrace();
        }

    }
}
```

RELATED CONCEPTS

Inheritance and association
Access beans: overview
Access beans: Java bean wrappers
Access beans: copy helpers
Access beans: rowsets

RELATED TASKS

Adding enterprise beans with inheritance
Creating or editing associations
Creating or editing access beans

Notices

Note to U.S. Government Users Restricted Rights — Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:
INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OR CONDITIONS OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web

© Copyright IBM Corp. 1998, 2000

sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

Lab Director
IBM Canada Ltd.
1150 Eglinton Avenue East
Toronto, Ontario M3C 1H7
Canada

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples may include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. 1997, 2000. All rights reserved.

Programming interface information

Programming interface information is intended to help you create application software using this program.

General-use programming interfaces allow the customer to write application software that obtain the services of this program's tools.

However, this information may also contain diagnosis, modification, and tuning information. Diagnosis, modification and tuning information is provided to help you debug your application software.

Warning: Do not use this diagnosis, modification, and tuning information as a programming interface because it is subject to change.

© Copyright IBM Corp. 1998, 2000

Trademarks and service marks

The following terms are trademarks of International Business Machines Corporation in the United States, or other countries, or both:

- AIX
- AS/400
- DB2
- CICS
- CICS/ESA
- IBM
- IMS
- Language Environment
- MQSeries
- Network Station
- OS/2
- OS/390
- OS/400
- RS/6000
- S/390
- VisualAge
- VTAM
- WebSphere

Lotus, Lotus Notes and Domino are trademarks or registered trademarks of Lotus Development Corporation in the United States, or other countries, or both.

Tivoli Enterprise Console and Tivoli Module Designer are trademarks of Tivoli Systems Inc. in the United States, or other countries, or both.

Encina and DCE Encina Lightweight Client are trademarks of Transarc Corporation in the United States, or other countries, or both.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries.

ActiveX, Microsoft, SourceSafe, Visual C++, Visual SourceSafe, Windows, Windows NT, Win32, Win32s and the Windows logo are trademarks or registered trademarks of Microsoft Corporation in the United States, or other countries, or both.

UNIX is a registered trademark in the United States and other countries licensed exclusively through X/Open Company Limited.

Intel and Pentium are trademarks of Intel Corporation in the United States, or other countries, or both.

Other company, product, and service names, which may be denoted by a double asterisk(**), may be trademarks or service marks of others.

© Copyright IBM Corp. 1998, 2000