COMPUWARE CORPORATION
Systems Software Division — Distributed Products
31440 Northwestern, Farmington Hills, MI USA 48334-2564
Technical Support Hotline: 1-800-538-7822

Enterprise Java Testing with TestPartner and VBA using Sun Microsystems' Client Access Services COM Bridge

Compuware Corporation Technical Whitepaper

Introduction

Enterprise Java Beans

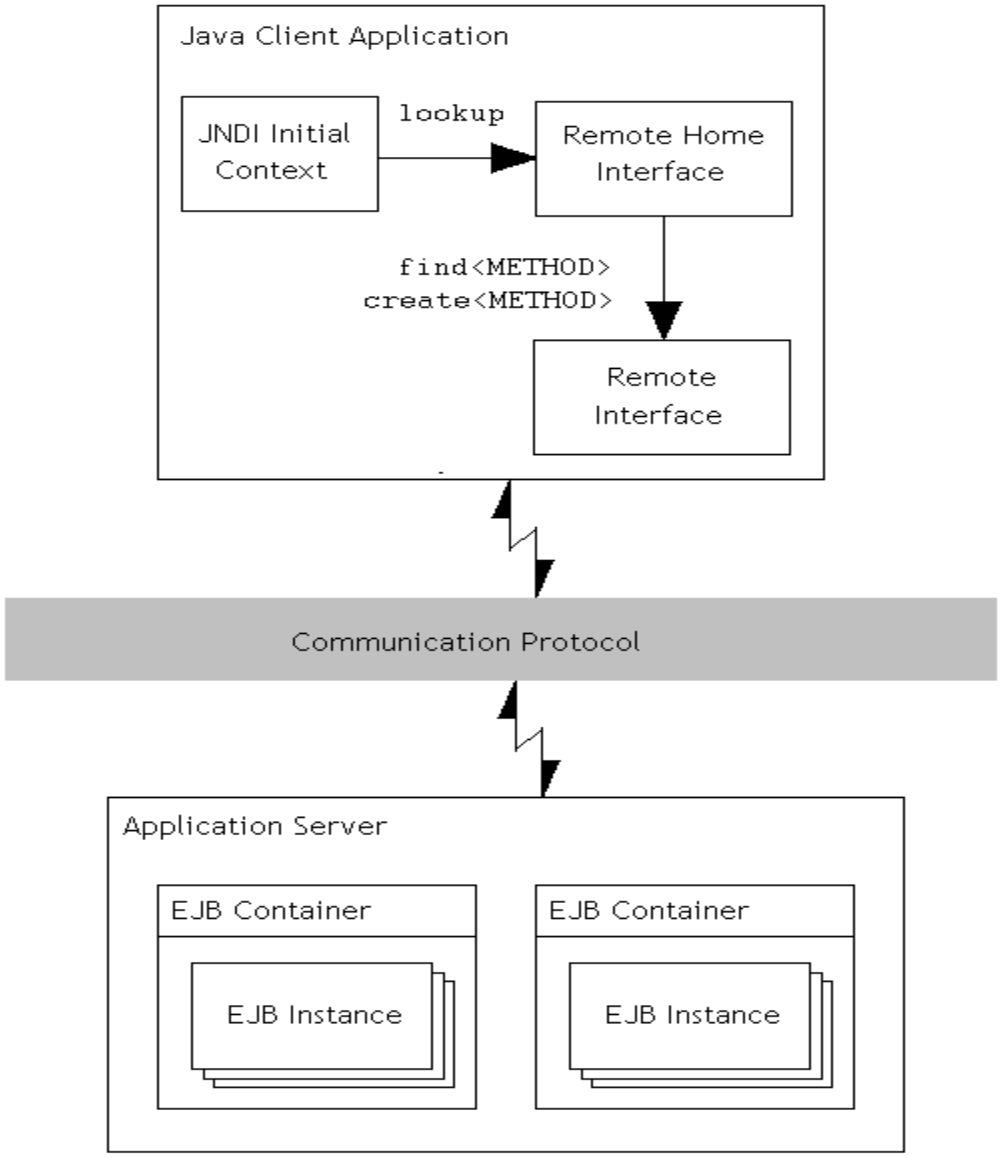
The Enterprise Java Beans architecture is the specification for server-side enterprise components developed using Sun Microsystems' Java programming language. It is a standard rather than a software architecture.

Enterprise Java Beans components (EJBs) are instances of Java classes that implement particular interfaces specified in the EJB Specification available for download from the Java website. The specification mandates that EJBs can run on any application server that can provide EJB containers that conform to the specification. EJB containers host running EJB instances providing them with a standard execution environment across all compliant servers.

It is also the responsibility of the application server to provide a client view of running EJBs that are intended for use by Java client applications running on other machines. This client view consists of an instance of a Java class that implements the **javax.ejb.EJBObject** interface. This instance comprises the EJB's Remote Interface. Clients interact with EJBs through their Remote Interfaces, with the Remote Interface-implementing class performing communication with the running EJB instance in a way that is dependent on the application server and communication protocol used.

In a similar manner, there is a standard Java interface that is used to create or find particular EJB instances. For each EJB there is a corresponding EJB Home. The application server container that hosts an EJB provides a class that implements the EJB Home's Remote Home Interface. This Remote Home implementation is made available to clients via the Java Naming and Directory Interface (JNDI). When a client application wishes to make use of a particular EJB, it must first perform a JNDI lookup to acquire the EJB's Remote Home interface. This allows the client to call methods on the Remote Home to find a particular instance, or set of instances, of the EJB, create new EJB instances, and remove EJB instances.

This graphic represents an overview of this interaction between EJBs and Java client applications:



The J2EE Client Access Services COM Bridge

In order to access EJB components a client application should be written in Java. Even Web clients that view HTML output generated by EJB components usually receive that HTML indirectly via an intermediate Java servlet.

A client application must be written in Java to richly and directly access EJB components. Even so, the means by which that application communicates with those components can differ among application servers for the same EJB application.

Sun Microsystems addressed these issues by requiring that Remote Method Invocation over the Internet Inter-Orb Protocol (**RMI-IIOP**) be supported as the standard EJB wire protocol by all application servers. It also addressed these issues in the form of the J2EE Client Access Services COM Bridge. This is a

collection of COM objects that allow any COM-enabled application to create and interact with Java objects running in a Java Virtual Machine (JVM). The COM Bridge creates a new Java Virtual Machine within the COM client process, and allows the JVM version to be selected from those installed on the client machine. At its lowest level, this allows such applications to perform the same operations with which a Java client application would create the Initial JNDI Context. This Initial Context can then be used via the COM Bridge to look up the EJB Home interface of the desired EJB, and obtain EJB instances.

In addition, the COM Bridge provides COM objects that build on the low level objects to provide Enterprise Services modules. These are application-server specific and are used to partially automate the process of connecting to the application server and looking up EJB Home interfaces.

Support for scripting systems is provided by COM objects. These are designed to compliment such systems, and to overcome their limitations, with tools capable of generating type libraries for any Java class. This type library information enables IntelliSense features when writing scripts that manipulate Java objects exposed by the COM Bridge.

The remainder of this document describes the requirements and procedures involved in testing Enterprise Java Beans components from TestPartner VBA scripts using the COM Bridge. It concludes with a worked example that illustrates this process using a script to test one of the sample applications provided with the COM Bridge - an EJB representing a bank account. This runs on the J2EE Reference Implementation application server.

Preparation and Planning

Server and Application Discovery

The first stage in preparing to use the COM Bridge is to discover the type of server on which the target application runs. This will determine if one of the Enterprise Services modules provided with the COM Bridge can be used, or if it will be necessary to work directly with the COM Bridge itself. Enterprise Services modules are provided for the following application servers:

Sun Microsystems™ Inc.'s J2EE™ SDK versions 1.2.1 and 1.3
iPlanet's iPlanet Application Server version 6.0 SP2
BEA's WebLogic Server versions 5.1 and 6.0
IBM's WebSphere Application Server version 3.5SilverStream's SilverStream Application Server version 3.7

Application Server Vendors implementing the J2EE platform can use any chosen protocols and connection schemes. Connection requirements for an application will differ from server to server. However, the following information is typically required in all cases:

The *Provider URL* of the application or server. For example, the J2EE Reference implementation uses RMI-IIOP to communicate with clients, and therefore expects provider URLs of the form: iiop://<hostname>:<port>.

For applications that require authentication, a *Security Principal* and *Security Credentials* is required. In some cases this may be a simple username/password combination. In others, such as Netscape's IPlanet Application Server, instances of specific Java classes may be required.

The Java Naming and Directory Interface (JNDI) name to use when looking up the Remote Home interface of each EJB to be tested

The full class name of the Remote Home interface of each EJB to be tested

Use the documentation supplied with the COM Bridge's Enterprise Services modules to determine the connection requirements for the type of server you are testing against. Contact the server administrator and request this information for the application you wish to test. It is required before you can begin testing the application using the COM Bridge and VBA.

Determining Client Requirements

Due to the COM Bridge's use of a Java Virtual Machine hosted within the client process to access EJB applications, there are a number of requirements that must be met by the client machine before testing can take place.

While any recent version of the Java Virtual Machine and Java Runtime Environment can be used to access EJB applications, some Application Server Vendors may mandate that a particular VM of a specific version be used to access their servers. While some of these requirements are expected, such as recommending the latest version of the JVM to access applications deployed to the 1.3 J2EE Reference

Implementation, others may not be. It is important to determine which JVM is preferred or required before testing. Otherwise, unexpected behavior may result.

You should ensure that the correct JVM at the correct version is installed on the same machine as the COM Bridge. Note that you may need to install more than one JVM if you plan on testing more than one application that requires a specific JVM. The JVM used by the COM Bridge can be specified when it is started (See "Creating JVM Configuration Files")

When an Enterprise Java application is deployed to a particular server, the deployment tool also generates class files that allow access by Java clients. These class files are typically packaged as JAR (Java Archive) files. For each client-accessible EJB in the application, the JAR file contains the following:

The *Remote Home* interface of the EJB. This interface provides the client application with the means to create or find instances of the EJB.

The *Remote* interface of the EJB itself. This interface exposes the business logic methods of the EJB and allows the client to interact with the EJB running on the application server.

You should determine what client class or JAR files are necessary to access the application to be tested and ensure that they are present on the same client machine as the COM Bridge. Without the client class files, the JVM that the COM Bridge creates cannot access the enterprise application.

The COM Bridge exposes objects running in its JVM by means of a JavaProxy COM object, which is based on the IDispatch interface. This allows the JavaProxy to dynamically represent Java objects of any class and also allows clients such as VBA to use late binding. However, it also makes the JavaProxy object appear to VBA to be of type 'Object'. This is because VBA does not have any means of accessing or interpreting the Java class information for the Java object itself.
To overcome this limitation, tools accompany the COM Bridge that supply type information about Java objects using the same mechanism as all other COM objects – type libraries. The GenTypeLib tool is capable of building and registering type library information for one or more Java classes. For more information on both the GenTypeLib and RegTypeLib tools, see "Generating and Registering Type Library Information". Once registered, a generated type library enables IDE features such as IntelliSense and automatic statement completion when dealing with Java objects created by the COM Bridge, making those objects much easier to work with.

You should determine which client Java classes you will need to use from VBA during testing of your enterprise applications. These are the classes that should have type library information generated for them. Typically, you should consider generating type library information for the following:

The Remote Home interface of each EJB that you will be testing

The Remote interface of each EJB that you will be testing

Any utility classes needed to construct parameters to pass to EJB methods, or needed to interpret return values. This includes *java.util.Collection* and *java.util.Iterator* for working with methods that return multiple objects; *java.util.Date*, and *java.lang.System*. Note that the class *java.lang.String* is already mapped directly to the Visual Basic String type.

Any classes related to the specific connection requirements of the application server hosting the application to be tested.

Since testing requirements for the COM Bridge differ between application servers, additional steps may be necessary before using the COM Bridge to test the target application. For example, the COMBRIDGE_HOME environment variable should be set to the installation directory of the COM Bridge, and it may be necessary to set the JAVAHOME environment variable specifically for the target application. You should consult the Enterprise Services documentation supplied with the COM Bridge to determine what additional requirements are imposed when testing against the target application server.

Installing Java Runtimes

The Java Virtual Machines required for testing should be installed before installation of the COM Bridge itself. During COM Bridge installation, an installed VM must be designated as the default JVM to instantiate if the COM Bridge is started without a JVM specified. Since it is desirable to specify the most recent JVM for this purpose, and since JVMs should be installed in order from least to most recent, it is recommended that you install required JVMs before installing the COM Bridge.

Java Virtual Machines suitable for use with the COM Bridge are those provided as part of Sun's Java Development Kits (JDKs) or Java Runtime Environments (JREs) that include the **jvm.dll** library file.

Installing the Client Access Services COM Bridge

To install the COM Bridge specify the installation location, and select a default JVM.

When selecting the default JVM, the required JVM may not appear in the list of JVMs found by the install program. Browse for the jvm.dll file of the required JVM. Depending on the version of the JDK or JRE, and also on the testing requirements, this file may be in a number of locations. Within the 'JRE' subdirectory of the JVM installation directory, appropriate jvm.dll libraries can typically be found in the 'classic', 'client', or 'hotspot' subdirectories. For example, if Java 2, Standard Edition v1.4 was installed in the 'C:\j2dsk1.4' directory, then candidate JVMs could be found in the 'C:\j2dsk1.4\JRE\classic' or 'C:\j2sdk1.4\JRE\client' directories.

The default JVM used by the COM Bridge can be changed after installation by running the provided 'COM Bridge Configuration' program.

Generating and Registering Type Library Information

The COM Bridge provides two tools with which to generate and register type library information for client Java classes: GenTypeLib and RegTypeLib. GenTypeLib is responsible for building and registering type library information from Java classes, whereas RegTypeLib is capable of registering type library information previously generated by GenTypeLib. Both tools can be found in the 'bin' subdirectory of the COM Bridge installation directory.

Use GenTypeLib as follows:

```
GenTypeLib –javaclass <class name>
```

This generates a type library containing information about the single java class specified, which is located using the current classpath.

Multiple –javaclass <class name> pairs can be specified. The '-lib' switch must be used to specify the name of the overall type library. The JAR file containing the class files can also be specified using the '-jar' switch.

The J2EE CAS COM Bridge Reference Manual provided as part of the COM Bridge documentation contains detailed information about both GenTypeLib and RegTypeLib.

Creating JVM Configuration Files

The COM Bridge allows control over a number of startup parameters, including the installed JVM to use, the system and boot classpaths, and startup options for the JVM itself. These parameters are made available as properties of the **JvmControl** ActiveX object, which allows the COM Bridge to be embedded in any ActiveX control container. The property sheets of the JvmControl allow design-time configuration of the COM Bridge startup parameters. However, the JvmControl is also useful for scripting purposes. This is due to the inclusion of the 'JVM Configuration Editor' utility, which uses the same property sheets as the JvmControl to modify startup options, but saves the result to a file. At runtime, a VBA script can create a JvmControl instance and instruct it to load the configuration information contained in the file. This is used by the JvmControl when starting the COM Bridge JVM.

A shortcut to the JVM Configuration Editor is added during COM Bridge installation.

It is strongly recommended that you create a configuration file for each application under test. This will simplify starting the COM Bridge from VBA, and also build a reusable collection of configurations.

Preparation and Planning Checklist

Before installing the COM Bridge, you should

Discover the following connection requirements of the target application and the server that hosts it:

- Appropriate enterprise services module.
- Provider URL.
- Security principal and credentials.
- JNDI names.
- Remote Home class names.
- Any other server- or application-specific requirements.
- Install the necessary Java Virtual Machines.
- Determine which Java class files or JAR files are needed for the COM Bridge to communicate with the EJB application, and make sure these files are on the machine.

After installing the COM Bridge and **before testing**, you should

- Set the COMBRIDGE_HOME environment variable.
- Possibly set the JAVA_HOME environment variable, depending on the target application.
- Optionally create a JVM configuration that can be used to initialize the COM Bridge for testing the target application.
- Determine which client class files will be used from VBA. Generate and register type library information for these classes.
- Address any other application- or application-server-specific requirements.

Using the J2EE Client Access Services COM Bridge from VBA

Introduction to the Client Access Services COM Bridge COM Objects

This section outlines several of the most significant objects that comprise the COM Bridge. Detailed information these objects can be found in the J2EE CAS COM Bridge Reference Manual supplied with the COM Bridge.

The COM Bridge is built on a small number of core objects, namely the **JvmStarter**, **JavaVirtualMachine**, and **JavaProxy** objects.
It is usually unnecessary to work directly with these objects when testing against an application server supported by one of the Enterprise Services modules supplied with the COM Bridge. They are used indirectly through the high-level **JvmControl**, **JavaServices**, **ScriptingServices** and Enterprise Services objects.

The JvmStarter object is used to set startup options for the COM Bridge JVM, including system and boot classpath, system properties and JVM options. Once configured, Its **StartJVM** method creates the JVM in the client process. This method returns a reference to a JavaVirtualMachine object representing the newly started JVM.

The JavaVirtualMachine allows the client process, and therefore TestPartner VBA scripts, to interact directly with the running JVM. This provides the ability to load classes, create new objects, and invoke static methods. Objects instantiated in this way are returned to VBA as a **JavaProxy** object. The JavaServices and Enterprise Services modules provide higher-level means of instantiating Java objects.

JavaProxy objects are used by the COM Bridge to represent Java objects that exist within the COM Bridge JVM. The JavaProxy implements IDispatch and uses Java's reflection mechanism to invoke methods on the underlying Java object. This allows JavaProxy objects to dynamically represent objects of any Java class type. Type library information about Java classes, provided by the GenTypeLib tool, removes the need to work with JavaProxy objects as instances of IJavaProxy or as unknowns of type 'Object'. Returned JavaProxy objects can be assigned to variables of the correct type from the type library generated for the class of the underlying Java object. This allows JavaProxy objects to be treated as instances of a specific type that is fully supported by the IntelliSense features of the VBA code editor.

The JvmControl object provides essentially the same functionality as the JvmStarter object. However, as mentioned previously it is an ActiveX control, and also provides the ability to load and save JVM configurations. This feature simplifies the process of specifying JVM startup options and starting the JVM from VBA scripts. After calling the Load method to load a JVM configuration file, the **StartJVM** method of the JvmControl can be used to create and start the COM Bridge JVM. As for the JvmStarter, this method returns a JavaVirtualMachine object.

The JavaServices object provides a more convenient way of creating Java objects, finding classes, and invoking static methods than those provided by the JavaVirtualMachine object. In most cases, the JavaServices object reduces a multi-step process to one method call. For example, creating an object using the JavaVirtualMachine object requires calling the **GetSystemClassLoader** method, then using the returned class loader in a call to the **FindClass** method. Finally, the New method can be called with the found class as the first parameter. The same result can be achieved simply by calling the **JavaNew** method of the JavaServices object.

While the JavaServices object adds convenience, the ScriptingServices object addresses some of the limitations imposed by using the COM Bridge from a scripting language such as VBA. Automation clients

such as VBA are capable of dealing with JavaProxy objects only as instances of the IDispatch interface. The ScriptingServices object allows scripts to indirectly access methods of the IJavaProxy interface implemented by the JavaProxy object. These are the **Invoke**, **InvokeA**, and **GetJavaClass** methods. Support for miscellaneous Variant-related conversion operations is also provided to overcome VBA's inability to handle non-Variant arrays and the primitive Java type long.

For each supported application server there is a corresponding Enterprise Services module that handles the particulars of communicating with servers of a specific type. Each of these supports the **ProviderURL**, **SecurityPrincipal** and **SecurityCredentials** properties for specifying the connection details, and the **lookupEjbHome** method to query for the Remote Home interface of EJBs residing on the server. Each Enterprise Services module is contained within a DLL installed in the 'bin' or 'doc\appserver-connectivity-samples\bin' subdirectories of the COM Bridge installation directory. The DLL and COM Class names for each of these modules are summarized below:

Application Server	DLL	COM Class
Sun Microsystems J2EE 1.2.1 and 1.3 Reference Implementation	J2ee-ri-services.dll	J2eeRiServices
iPlanet's iPlanet Application Server version 6.0 SP2	ias-services.dll	IasServices
BEA's WebLogic Server versions 5.1 and 6.0	weblogic-services.dll	WeblogicServices
IBM's WebSphere Application Server version 3.5	websphere-servcices.dll	WebSphereServices
SilverStream's SilverStream Application Server version 3.7	silverstream-services.dll	SilverStreamServices

Overview of the EJB Testing Process

The process of testing EJBs from TestPartner using VBA is as follows:

Creating a new script and adding COM Object references

After creating a new TestPartner VBA script that will become the EJB test script, it is necessary to add the appropriate references to the script to make use of the type library information available for the COM Bridge objects themselves, and also for the type libraries generated for Java classes during the preparation and planning process. References to the following type libraries should be added:

J2EE CAS Com Bridge 1.0 Type Library
J2EECAS JvmControl 1.0 Type Library

J2EE CAS Java Services 1.0 Type Library
J2EE CAS Scripting Services 1.0 Type Library
The type library of the Enterprise Services module(s) that will be used in the script
The type libraries generated for the EJB Remote Home interface and the EJB Object interface of each EJB to be tested by the script.
The type libraries generated for classes of the Java objects that will be used in the script.

Starting the COM Bridge JVM

Before using the Enterprise Services object appropriate for the application server under test, the JVM used by the COM Bridge must be started using the JvmControl object. Starting the JVM using one object and accessing it via another works because the COM Bridge is only capable of creating one JVM in the client process at a time, which is shared by all COM Bridge components.
From the VBA script, starting the COM Bridge JVM is as simple as declaring a new instance of the JvmControl, loading the correct configuration file, then calling the JvmControl's StartJVM method. Without a configuration file, you must set properties of the JvmControl to the required values programmatically before calling its StartJVM method.

Another alternative is to place the JvmControl on a UserForm, set its properties at design time through its property sheets, then call its StartJVM method in the **UserForm_Initialize** event handler of the UserForm.

Connecting to the Application Server

Connecting to the application server initially involves creating an instance of one of the Enterprise Services COM classes. Once the ProviderURL property, and possibly the SecurityPrincipal and SecurityCredentials properties, of this object have been specified, the Enterprise Services object is ready for use. If security information is required it may be necessary to use a JavaServices object to create instances of Java objects to use as, and in the construction of, security principal and credentials.

Looking up EJB Remote Home interfaces

After the Enterprise Services object has been initialized, its **lookupEjbHome** method can be used to retrieve the Remote Home interfaces of EJBs under test. This method takes a JNDI name and the full class name of the Remote Home interface. The returned object should be assigned (using the VB 'Set' keyword) to a variable of the correct type from the type library generated for the EJB Home interface.

Performing EJB find and create operations

Once the Home interface has been obtained, its find and create methods can be invoked to obtain references to the desired EJB objects. These include methods such as **findByPrimaryKey** and **findAll**. These methods may return a single instance or multiple instances contained in a Collection. Objects returned from these methods should be assigned (using the VB 'Set' keyword) to variables that are either of type java_util_Collection or of the correct type from the type library generated for the EJB class. (The java_util_Collection type will only be available to the script if type library information has been generated for the Java class java.util.Collection and a reference added for this type library. If you use

java.util.Collection, it is likely that java.util.Iterator should also have type library information generated for it, since iterators are the most common method used to access Collection elements).

Exercising EJB business logic methods

An EJB object obtained via its Home interface and assigned to a variable of the correct type can be manipulated in the same manner as any other automation object created in a VBA script. Methods can be invoked and properties can be set and retrieved, assisted by the IntelliSense features provided by the type library reference for the EJB's class. Again, it may be necessary to make use of the JavaServices, and possibly ScriptingServices, objects in order to construct parameters to EJB methods or interpret return values.

Shutting down the COM Bridge JVM

When the script has completed testing, it should attempt to shut down the JVM created by the COM Bridge. Shutting down the JVM is a matter of calling the **GetJvmStarter** method of the JvmControl used to start it, then calling the **StopJvm** method of the JvmStarter.
The documentation provided with the COM Bridge states that this method "does not work reliably with all JVMs, and may produce unexpected results". However, the attempt should still be made, since only one JVM can be created within the client process at a time. This requires a TestPartner VBA script that creates a JVM of a particular version to shut down that JVM before a JVM of a different version can be created within the TestPartner process.

Worked Example: The Bank Account EJB

The following section demonstrates the preparation and planning stage and the EJB testing process for one of the examples supplied with the COM Bridge (Example 6). It involves using the COM Bridge to access a simple EJB object representing a bank account. Instances of the EJB are hosted on the J2EE Reference Implementation application server.

This worked example will lead you through the initial setup of the J2EE Reference Implementation application server and the EJB application. It will then illustrate the various stages of the preparation and planning phase, and how the testing process applies in this specific instance. It only makes use of the EJB application supplied as part of the COM Bridge's Example 6 example. It does not use any of the batch files or VB code supplied. The batch files are not used because they automate many of the steps involved in the preparation and planning phase, and so would not serve as a useful demonstration of how that phase should proceed. VBA code will be added to a new script at each point in the testing process, resulting in a script that follows the specified EJB testing process.

The complete TestPartner VBA script for this example is given at the end of this section.

Setup of the Bank Account EJB Example

This consists of the following steps:

1. Installing required components
2. Configuring the bank account example database
3. Deploying the bank account example application

Installing Required Components

Since the J2EE RI application server is running locally, and requires the installation of the Java 2 SDK 1.2.2 or later to operate, it is necessary to deviate slightly from the preparation and planning checklist, which shows installation of JVMs as occurring after discovery of connection requirements. In this example, the JVM supplied with the Java 2 SDK is required by the J2EE RI application server, and is installed as part of this setup operation. The final setup stages require COM Bridge installation to have already occurred, since they use materials provided as part of Example 6.

Therefore, the installation order for this example is as follows

1. Install the Java 2 SDK version 1.2.2 or later
2. Install the J2EE SDK
3. Install the COM Bridge

After installation, you should set the J2EE_HOME environment variable to reflect the directory where the J2EE SDK was installed, and the JAVA_HOME environment to reflect where the Java 2 SDK was installed. The COMBRIDGE_HOME environment variable should also be appropriately set.

Configuring the Bank Account Example Database

The bank account example application depends on a database containing account information in order to operate. This database must be configured before the application can be used. To accomplish this, perform the following steps:

1. Start the CloudScape database server that is provided as part of the J2EE Reference Implementation by opening a new command prompt and typing:

```
cd /d %J2EE_HOME%\bin
cloudscape -start
```

2. Once CloudScape is running, create the bank account example database by opening another command prompt and typing:

```
SET CLASSPATH=%J2EE_HOME%\lib\system\cloudutil.jar;
⇨%CLASSPATH%
cd /d %COMBRIDGE_HOME%\doc\guide\examples\bin
cloudutil ..\sql\create.sql
cloudutil ..\sql\insert.sql
```

Deploying the Bank Account Example Application

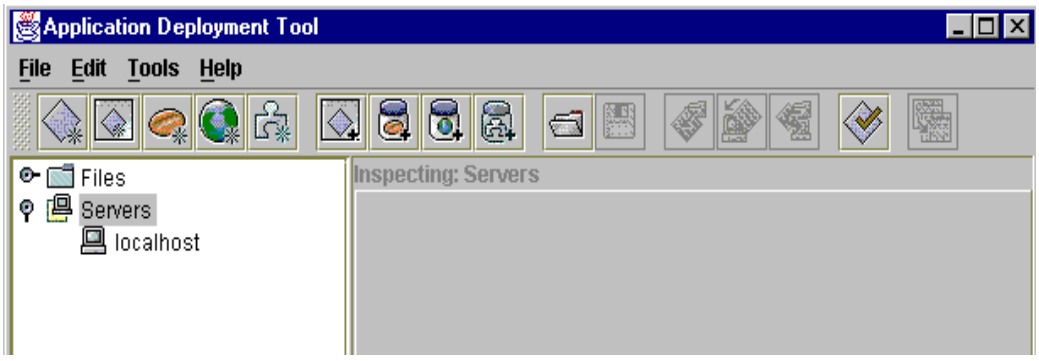
1. With the CloudScape database server still running, start the J2EE RI application server by typing:

```
cd /d %J2EE_HOME%\bin
j2ee -verbose
```

2. Start the J2EE RI Deployment Tool by opening a new command prompt and typing:

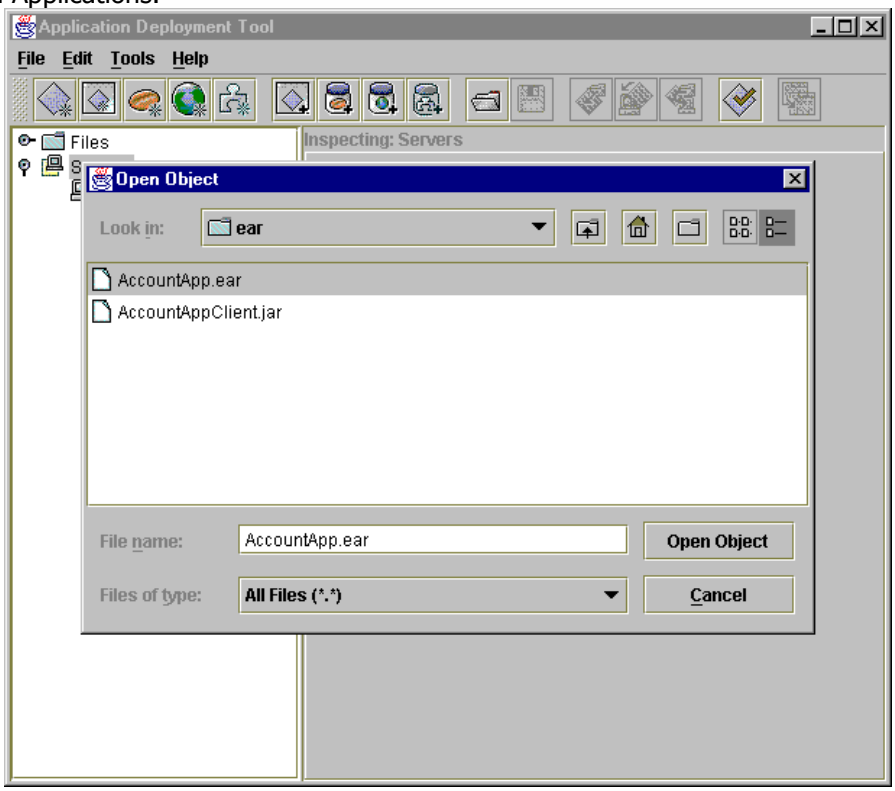
```
cd /d %J2EE_HOME%\bin
deploytool
```

3. When the Deployment Tool window appears, verify that 'localhost' is listed under the 'Servers' branch of the tree:

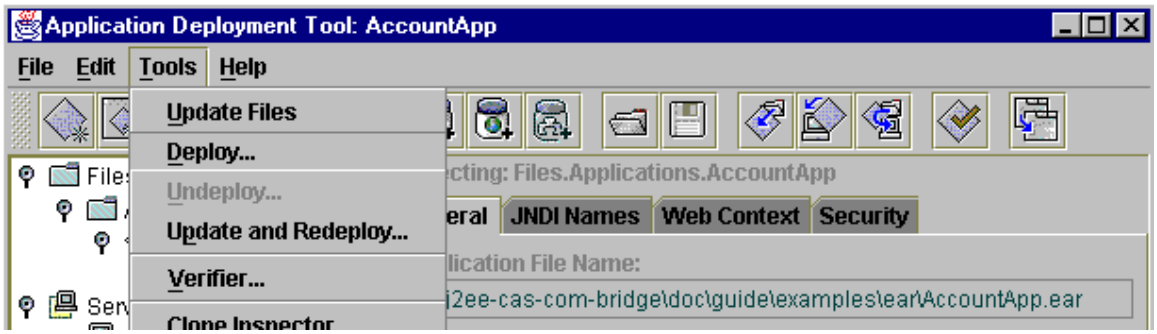


4. Select **Open** from the **File** menu, and browse to the COM Bridge installation directory, for example C:\j2ee-cas-com-bridge. Then browse to the 'doc\guide\examples\ear' subdirectory,

select the AccountApp.ear file and select **Open Object**. A new branch of the tree should appear labeled Applications.

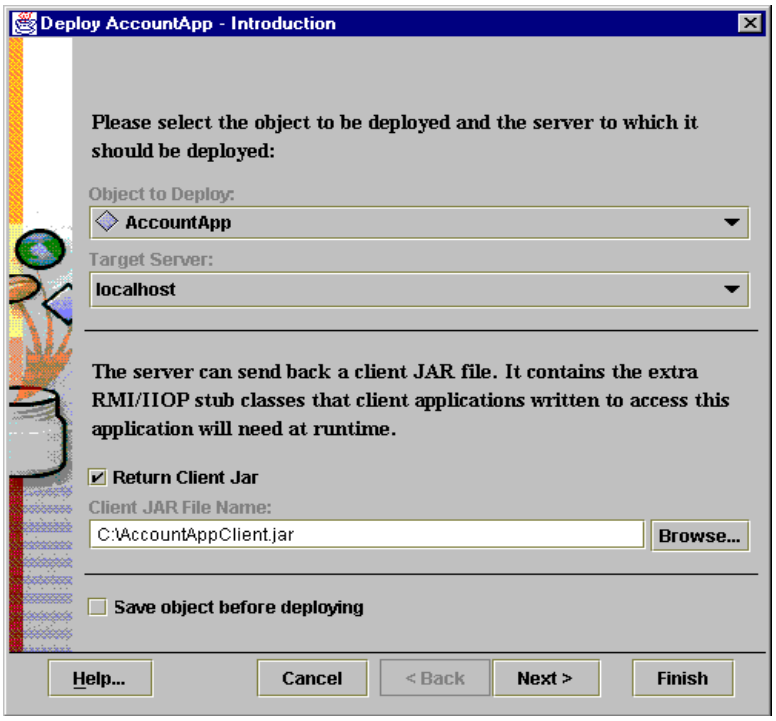


5. Make sure that the AccountApp child of this branch is selected, then select **Deploy** from the Tools menu to start the deployment process:



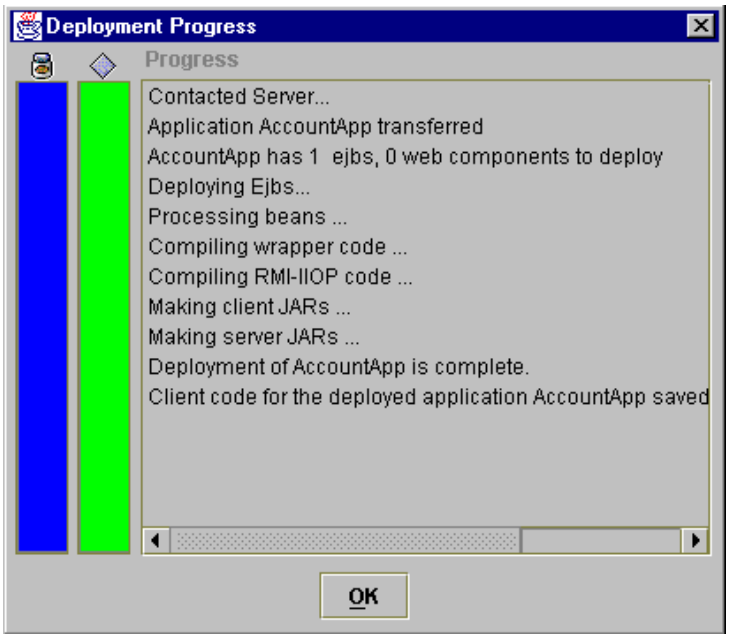
6. In the Introduction screen, verify that AccountApp and localhost are the selected application and host. Check Return Client JAR and change the Client JAR File Name to something simple such as C:\AccountAppClient.jar

7. Deselect Save object before deploying. Note that the client JAR file returned by this deployment process will be required by the COM Bridge in order to test the example application.



8. Select **Next** to proceed to the JNDI Names deployment properties page. Verify that the Application table contains an EJB component called AccountBean with a JNDI name of`MyAccount, and that there is a resource with the JNDI name jdbc/CloudScape.

9. Select **Next** to proceed to the deployment review page. Select **Finish** to finalize the deployment. You can monitor the deployment progress in both the deployment tool and the command prompt in which the J2EE RI server is running. When complete, this dialog box appears:



Select **OK** to close the dialog. You have successfully deployed the bank account example EJB application.

Using the Preparation and Planning Checklist for the Bank Account EJB Example Application

Using the **Preparation and Planning Checklist** on page 8, the following are specific requirements for the Bank Account EJB example:

Appropriate enterprise services module
Since the example application is hosted on a J2EE Reference Implementation application server, the correct enterprise services module to use is the J2eeRIServices module.

Provider URL
The application server to which the example application is deployed is the local machine. Therefore the host part of the provider URL is 'localhost'. During startup, the J2EE RI server should have reported that it was starting the listen service on port 1050. Since the correct protocol for communicating with the J2EE RI server is the Internet Inter-Orb Protocol, the URL of the bank account example is iiop://localhost:1050.

Security principal and credentials
The bank account example does not require authentication. Therefore a security principal and credentials are not necessary.

JNDI names
The JNDI name specified during the deployment process is needed. This is the name given to the AccountBean EJB component, which by default is "MyAccount".

Remote Home class names

From examination of the client JAR file returned in the deployment phase, it can be seen that the name of the Remote Home class for the bank account EJB is account.AccountHome.

Install the necessary Virtual Machines
A JVM suitable for testing the bank account example EJB was installed during the setup process as part of the JDK 1.2.2 or later. This JVM should also be the designated COM Bridge default. Note that this is only the case because the J2EE RI application server is installed locally. It is typically necessary to install an appropriate JVM before testing is possible.

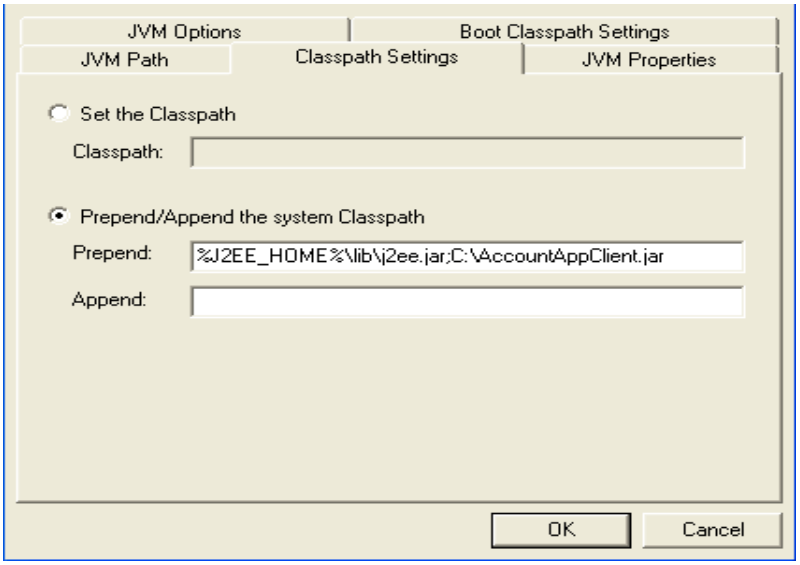
Determine the Java .class or JAR files necessary for communication with the EJB application
For the bank account example EJB, the client JAR file returned during the deployment process is the primary requirement. However, the J2EE SDK classes are also necessary. Therefore you should know the location of the client JAR file (given as C:\AccountAppClient.jar in the example deployment) and the j2ee.jar file, which is installed in the %J2EE_HOME%\lib\ directory. For example, if the J2EE SDK had been installed in the C:\j2sdkee directory, the path to the J2EE JAR file would be C:\j2sdkee\lib\j2ee.jar

Set the COMBRIDGE_HOME environment variable
This needs to be set to denote the directory where the COM Bridge was installed during the initial setup process. For example, C:\j2ee-cas-com-bridge.

Possibly set the JAVA_HOME environment variable, depending on the target application
This has already been done, since the J2EE SDK requires that the JAVA_HOME environment variable be set before the J2EE RI application server will start.

Optionally create a JVM configuration that can be used to initialize the COM Bridge
The bank account is a relatively simple example, and so a configuration file is not as helpful as it would be for a more complex application. The JVM required to run the J2EE RI application server is suitable for client use. The bank account example does not require any additional client-side JVM options. You must tailor the classpath. This must include references to the required client class files, which have previously been identified as the client JAR file returned during deployment and the J2EE SDK JAR file.
If a configuration file is used, the location of these files should be prepended to its classpath.

Start the JVM Configuration Editor using the Start menu shortcut installed by the COM Bridge. Click 'Edit' to bring up the property sheets for the configuration. Ensure that the 'JVM Path' tab is set to 'Use Com-Bridge default' and the 'Classpath Settings' tab is similar to the following:



Click **OK** to confirm these settings. Then, on the JVM Configuration Editor main window, review the settings, then click **Save As** to save the configuration to an appropriate filename such as C:\AccountAppConfig.jctl. If a configuration file is not used, these files should be prepended to the classpath using the appropriate methods of the JvmControl object.

Determine which client class files will be used from VBA. Generate and register type library information for these classes.
For the bank account example, the Remote Home and Remote interfaces are the **account**.**AccountHome** and **account**.**Account** classes, respectively. The example VBA script will make use of the find methods of the Home interface, which may return more than one result. Therefore, type library information will be necessary for the **java**.**util**.**Collection** and **java**.**util**.**Iterator** classes. However, since the methods of the Home and Remote interfaces of the bank account EJB take either no parameters or parameters that map directly onto VBA types, there are no utility classes for which type library information need be generated. Similarly, since the example application does not require authentication, no type libraries need be generated for authentication-related classes.

To GenTypeLib tool must be able to find the Java classes for which to generate type library information. Therefore, the location of the bank account example client JAR file must be included in the classpath before running the tool.

For example, if the client JAR file was returned during deployment as C:\AccountAppClient.jar, then the necessary type library information can be generated by opening a new command prompt and typing the following:

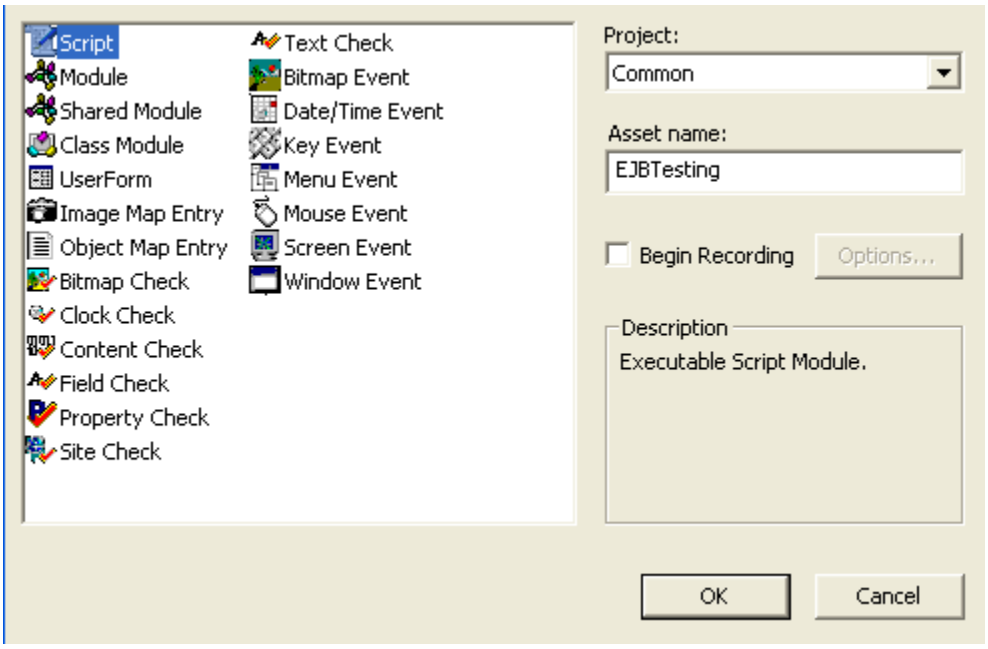
```
cd /d %COMBRIDGE_HOME%\bin
SET CLASSPATH=C:\AccountAppClient.jar;%CLASSPATH%
GenTypeLib -javaclass account.AccountHome
GenTypeLib -javaclass account.Account
GenTypeLib -javaclass java.util.Collection
GenTypeLib -javaclass java.util.Iterator
```

Address any other application- or application-server-specific requirements.
These requirements have already been addressed during the initial setup process.

Following the EJB Testing Process for the Bank Account EJB Example Application

Create a new script and add COM object references

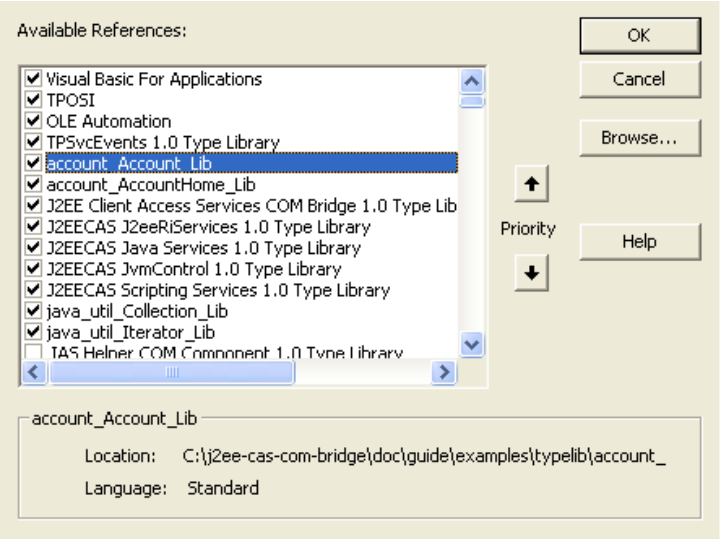
1. After logging in to TestPartner select **New** from the **File** menu to display the New Asset dialog.
2. Select **Script** in the asset types list.
3. Call the script EJBTesting and set the options as shown below:



4. Select **References** from the Tools Menu.

5. Ensure that each of the following is checked in the Available References dialog:

   account_Account_Lib
   account_Account_Lib
   J2EE Client Access Services COM Bridge 1.0 Type Library
   J2EECAS J2eeRiServices 1.0 Type library
   2EECAS JavaServices 1.0 Type Library
   J2EECAS JvmControl 1.0 Type Library
   2EECAS ScriptingServices 1.0 Type Library
   java_util_Collection_Lib
   java_util_Iterator



Starting the COM Bridge JVM

As previously specified, starting the COM Bridge JVM is a matter of declaring a new JvmControl instance, configuring it by either loading a configuration file or setting properties, then calling its StartJvm method. If you created a configuration file while following the preparation and planning checklist for the bank account EJB example, the VBA code you need to add to the new script will be similar to the following:

```
Dim jvmcontrol As New JvmControl

jvmcontrol.Load "C:\AccountAppConfig.jctl"
jvmcontrol.StartJvm
```

C:\AccountAppConfig.jctl should be replaced by the name of the configuration file. Otherwise, the required VBA code will be similar to this:

```
Dim jvmcontrol As New JvmControl

jvmcontrol.PrependClasspath =
"%J2EE_HOME%\lib\j2ee.jar;C:\AccountAppClient.jar"
jvmcontrol.StartJvm
```

C:\AccountAppClient.jar should be replaced by the filename used to save the client JAR file returned during the deployment process.

Connecting to the Application Server

From following the preparation and planning checklist, it is known that the example application is hosted by a J2EE Reference Implementation application server and is accessible using the J2eeRiServices class. Since the application is known to not require authentication, the Provider URL is the only other information required to connect to the application server. With or without a configuration file, the VBA code to add looks the same:

```
Dim j2ee As New J2eeRiServices
j2ee.ProviderURL = "iiop://localhost:1050"
```

Looking up EJB Remote Home interfaces

As previously mentioned, this is as simple as calling the lookupEjbHome method of the J2eeRiServices object. To do this, the JNDI name and full classname are needed. These are known to be "MyAccount" and "account.AccountHome", respectively. The value returned from the call to lookupEjbHome should be assigned (using the 'Set' keyword) to a variable of type account_AccountHome from the account_AccountHome_Lib type library. This will provide IntelliSense when working with the Remote Home interface. The necessary VBA code is as follows:

```
Dim acchome As account_AccountHome

Set acchome = j2ee.LookupEjbHome("MyAccount", "account.AccountHome")
```

Performing EJB Find and Create Operations

In this example, the findAll method of the Home interface will be used to return a collection containing all the bank account EJBs present based on the data in the CloudScape database. Other findBy methods of the Home interface, such as findByPrimaryKey, allow more selective queries. The collection object, which is an instance of the java.util.Collection class, is assigned (using the 'Set' keyword) to a variable of type java_util_Collection from the java_util_Collection_Lib library, to enable IntelliSense.
From the collection, an iterator can be obtained, which then allows access to the Remote interface of each EJB in turn. After being assigned to a variable of type java_util_Iterator from the java_util_Iterator_Lib type library, the iterator's hasNext and Next methods can be used to cycle through all the EJBs present in the collection. Assigning each in turn to a variable of type account_Account enables IntelliSense while working with the Account object.

```
Dim coll As java_util_Collection
Dim iter As java_util_Iterator
Dim acc As account_Account

Set coll = acchome.findAll
Set iter = coll.iterator

While iter.hasNext
Set acc = iter.Next
…
      < Exercise EJB Business Logic Methods >
      …
Wend
```

Exercising EJB Business Logic Methods

Once the EJB has been assigned to an account_Account variable, it can be manipulated just like any other VBA object. The following code illustrates the process of building a string detailing the bank account by accessing the properties of the account_Account object, then displaying that description string in a TestPartner VBA message box:

```
Dim str As String

str = "Account ID: " & acc.AccountId
str = str & vbCrLf
str = str & "Account Holder: " & acc.FirstName
str = str & " " & acc.LastName
str = str & vbCrLf
str = str & "Balance: " & acc.Balance
```

```
str = str & vbCrLf
str = str & "Credit Limit: " & acc.CreditLimit
str = str & vbCrLf
MsgBox str
```

Shutting Down the COM Bridge JVM

The attempt to shut down the COM Bridge is made by obtaining the JvmControl's underlying JvmStarter object, then calling its StopJvm method, as shown below:

```
Dim starter As JvmStarter
Set starter = jvmcontrol.GetJvmStarter
starter.StopJvm
```

or, alternatively:

```
jvmcontrol.GetJvmStarter.StopJVM
```

Complete Code Listing for the Bank Account EJB Example Test Script

```
Sub Main()

    Dim jvmcontrol As New JvmControl
    Dim starter As JvmStarter
    Dim j2ee As J2eeRiServices
    Dim coll As java_util_Collection
    Dim iter As java_util_Iterator
    Dim acchome As account_AccountHome
    Dim acc As account_Account

    Dim str As String

    jvmcontrol.PrependClasspath =
    "%J2EE_HOME%\lib\j2ee.jar;C:\AccountAppClient.jar"

    jvmcontrol.StartJvm

    Set j2ee = New J2eeRiServices
    j2ee.ProviderURL = "iiop://localhost:1050"

    Set acchome = j2ee.LookupEjbHome("MyAccount",
    "account.AccountHome")

    Set coll = acchome.findAll
    Set iter = coll.iterator

    While iter.hasNext
        Set acc = iter.Next

        str = "Account ID: " & acc.AccountId
        str = str & vbCrLf
        str = str & "Account Holder: " & acc.FirstName
        str = str & " " & acc.LastName
        str = str & vbCrLf
        str = str & "Balance: " & acc.Balance
        str = str & vbCrLf
```

```
        str = str & "Credit Limit: " & acc.CreditLimit
        str = str & vbCrLf

        MsgBox str
    Wend

    jvmcontrol.GetJvmStarter.StopJvm

End Sub
```