Java clients -- table of contents

Development

Administration

4.7: Java Clients

In a traditional client server environment, the client requests a service and theserver fulfills the request. A single server is used by multiple clients.Clients can also access several different servers. This model persists for Java clients except now these requests make use of a client's runtime environment.

Prior to *J2EE* (Java ™ 2 Platform Enterprise Edition), a typical Web-based client application consisted of the following model:

```
browser (HTML file) -> servlet -> EJB
```

In this model, the client application requires a servlet to communicate with the Enterprise Java Bean (EJB), and the servlet must reside on the same machineas WebSphere Application Server.

With version 4.0, Java application clients cannow consist of the following models:

- Applet client
- J2EE application client
- Java thin application client

In the ***Applet client*** model, a Java applet is embedded in a HyperText Markup Language(HTML) document residing on a client machine that is remote from WebSphere Application Server.With this type of client, the user accesses an EJB in WebSphere Application Serverthrough the java applet in the HTML document.

The ***J2EE application client*** is a Java applicationprogram that accesses EJBs, JDBC databases, and Java Message Service message queues. The *J2EE application client* program runs on client machines. This program follows the same Java programming model as other Java programs; however, the*J2EE application client* depends on the application client runtime to configure its execution environment, and it uses the JNDI name space to access resources.

The ***Java thin application client*** provides a light-weight Java client programming model. This client is best suited for use in situations where a Java client application exists but the application must be enhanced to make use of EJBs, or where the client application requires a thinner, more light-weight environment than the one offered by the J2EE application client.

4.7.1: Applet client programming model

The Java Applet client provides a browser-based Java runtime that is capable of interacting with EJBs directly instead of indirectly through a servlet.

This client is designed to support those users who want a browser-based Java client application programming environment that provides a richer and more robust environment than the one offered by the`Applet->Servlet->EJB` model.

The programming model for this client is a cross between the Java application thin client and a servlet client. When accessing EJBs from this client, the EJB object references can be considered CORBA object references by the applet.

There is no tooling support for this client for developing, assembling or deploying the applet. You are responsible for developing the applet, generating the necessary client bindings for the EJBs and CORBA objects, and bundling these pieces together to be installed on or downloaded to the client machine. The Java applet client provides the necessary runtime to support communication between the client and the server.

Client side bindings are generated during the deployment phase of J2EE developmentusing the Application Assembly Tool. An applet can utilize these bindings, or you can generate client side bindings using the `rmic` command that is partof the IBM JDK installed with the WebSphere Application Server.

See article Packaging and distributing Java clients formore information.

The Applet client makes use of the RMI-IIOP protocol. The use of this protocol enables the applet to access EJB references and CORBA object references, but it is restricted in the usage of some supported CORBA services. If you combine the EJB and CORBA environments in one applet, you must understand the differences between the two programming models, and you must use and manage each appropriately.

The Applet client provides the runtime to support the J2EE Applet client. The J2EE Applet client does not have any tooling support for developing, assembling or deploying the Applet. The applet client runtime is provided through the use of the Java applet browser plug-in that is installed on the client machine using the WebSphere ApplicationServer Client CD.

Because the Applet client does not provide for a deployment descriptor, the Applet code cannot make use of the JNDI `java:/comp` lookup. The Applet must know the fully qualified location of the EJB in the JNDI namespace. For example, the `JNDI:java:/comp` allows lookup of enterprise java beans using a short name or a nickname such as:

```
java.lang.Object ejbHome = initialContext.lookup("java:/comp/env/ejb/MyEJBHome");MyEJBHome =
(MyEJBHome)javax.rmi.PortableRemoteObject.narrow(ejbHome, MyEJBHome.class);
```

But the code in an applet client must be more explicit:

```
java.lang.Object ejbHome =
initialContext.lookup("the/fully/qualified/path/to/actual/home/in/namespace/MyEJBHome");MyEJBHome =
(MyEJBHome)javax.rmi.PortableRemoteObject.narrow(ejbHome, MyEJBHome.class);
```

The Applet environment restricts accessing external resources from the browser runtime environment. Some of these resources can be made available to the Applet by setting the correct security policy settings in the JRE `java.policy` file. If given the correct set of permissions, the Applet client must explicitly create the connection to the resource using the appropriate API (JDBC, JMS, and others). This client does not perform any initialization of any of the services that the client applet may need. For instance, the client application is responsible for the initialization of the naming service, either through CosNaming or JNDI.

The following table describes the advantages and disadvantages of the *Applet client*:

Advantages	Disadvantages
• Light-weight client suitable for download. • Provides access to JNDI interfaces for EJB object resolution. • No distribution of the applet to the client machine required (performed through the browser)	• Designed for use in an intranet environment. • Lack of client runtime initialization of environment and services. • Lack of built-in support for local resource resolution and configuration. • Does not promote portability of client application code. • Requires a browser to be installed on the client machine.

4.7.1.1: Developing an Applet client

Unlike typical applets that reside on either Web servers or WebSphere ApplicationServers and can only communicate using the HTTP protocol, the WebSphere Applet clients are capable of communciating over the HTTP protocol and the RMI-IIOPprotocol. This additional capability gives the Applet direct access to enterprise java beans. As such, Applet clients have the following setup requirements:

- These clients are currently available on the Windows NT or Windows 2000 platforms.

  Support for additional platforms will be added in the near future.
  Check the prerequisites page for information on new platform support.

- They require one of these browsers:
  - Internet Explorer version 5.0+
  - Netscape Navigator 4.7+
- The browser must be installed before the client code is installed.
- The Applet client is installed from the *WebSphere Clients for Windows* CD by selecting option, "Java Application/Applet Thin Client."
- You must install the WebSphere Application Server Plug-in for the browser.Select option, "Java Application/Applet Thin Client," from the *WebSphere Clients for Windows* CD.
- From the WebSphere Application Server Java Plug-in Control, enter:

  ```
  -Djava.ext.dirs=\WebSphere\AppClient\lib
  ```

  1. The ***Java Run Time Parameters*** field issimilar to the command prompt when using command line options.Therefore, most options available from the command prompt (for example, -cp, classpath, and others),can be entered in this field as well.
  2. The control panel can be accessed from the **Start** menu.
     Click `start > control panel > WebSphere Java Plug-in Control`.
  3. The applet container is the Web browser and the Java Plug-in combination.You must first install the WebSphere Application Server Applet client so that the browser recognizes the IBM Java Plug-in.

Tag requirements

Standard applets require the HTML <APPLET> tag to identify the applet to the browser. The <APPLET> tag invokes the browser's Java Virtual Machine (JVM). Soan applet running on Internet Explorer will use Microsoft's JVM.

For applets to communicate with EJBs in the WebSphere Application Server environment, the<APPLET> tag must be replaced with these two new tags:

<OBJECT>
<EMBED>

Additionally, the `classid` and `type` attributes **cannot** be modified,and must be entered as described in the applet client example. Finally, the `codebase`attribute on the <OBJECT> tag must be excluded.

Do not confuse the `codebase` attribute on the<OBJECT> tag with the `codebase` attribute on the <PARAM> tag. Althoughboth are called `codebase`, they are separate entities.

The following code snippet illustrates the applet code. In this example, ***MyApplet.class***is the applet code, ***applet.jar*** is the file that contains the applet code, and ***EJB.jar*** is the file that contains the EJB code:

```
<OBJECT classid="clsid:8AE2D840-EC04-11D4-AC77-006094334AA9"
          width="600" height="500">
<PARAM NAME=CODE VALUE=MyAppletClass.class>
<PARAM NAME="archive" VALUE='Applet.jar, EJB.jar'>
<PARAM TYPE="application/x-java-applet;version=1.3">
<PARAM NAME="scriptable" VALUE="false">
<PARAM NAME="cache-option" VALUE="Plugin">
<PARAM NAME="cache-archive" VALUE="Applet.jar, EJB.jar">
<COMMENT>
<EMBED type="application/x-websphere-client" CODE=MyAppletClass.class
         ARCHIVE="Applet.jar, EJB.jar" WIDTH="600" HEIGHT="500"
         scriptable="false">
<NOEMBED>
 </COMMENT>
</NOEMBED>WebSphere Java Application/Applet Thin Client for
           Windows is required.
</EMBED>
 </OBJECT>
```

The value of the type attribute on the <EMBED> tag can also be:
`<EMBED type="application/x-websphere-client, version=4.0" ...`

Code requirements

The code used by an applet to talk to an EJB is the same as that used by a standaloneJava program or a servlet, except for one additional property called *java.naming.applet*.This property informs the `InitialContext` and the ORB that this client is anapplet rather than a standalone Java application or servlet.

When initializing an instance of the `InitialContext` class, the first two lines in this code snippet illustrate what both a standalone Java program and a servlet issue to specify the computer name, domain, and port. In this example, <`yourserver.yourdomain.com`> isthe computer name and domain where WebSphere Application Server resides, and `900` isthe configured port. After the bootstrap values (<yourserver.yourdomain.com>:900) are defined,the client to server communications occur within the underlying infrastructure. In addition to the first two lines, for applets, you must add the highlighted third line to your code. That line identifies this program as an applet:

```
prop.put(Context.INITIAL_CONTEXT_FACTORY,
"com.ibm.websphere.naming.WsnInitialContextFactory");
prop.put(Context.PROVIDER_URL, "iiop://<yourserver.yourdomain.com>:900");
prop.put("java.naming.applet", this);
```

Security requirements

When code is loaded, it is assigned "permissions" based on the security policy in effect.This policy, specifying which permissions are available for code from various locations,can be initialized from an external policy file.For each client, the *java.policy* file should be updated with the classes that the applet client needs to access, and with the ports on the host machines where it needs different permissions.

The following lines of code must be added to existing *java.policy* files.This code allows access to the required ports so that the applet client can communicate with an EJB.

In the example below, the `java.net.SocketPermission "localhost:1024--", "listen"`entry grants permision for Applets to open sockets for listening on the localhost for any portfrom 1024 to 65535. Port can be specified as a range of port numbers or a specific port. A port range specification of the form "N-", where N is a port number, signifies all ports numbered N and above. A specification of the form "-N" indicatesall ports numbered N and below.

The first SocketPermission statement grants permission to theclient to have ports opened for listening. The second grants permission toopen a port and make a connection to a host machine, which is your WebSphere Application Server.In this example, ***yourserver.yourcompany.com*** is the complete hostname of yourWebSphere Application Server:

```
permission java.util.PropertyPermission "server.root", "read";
permission java.util.PropertyPermission "*", "read,write";
permission java.io.FilePermission "traceSettingsFile", "read,write";
permission java.util.PropertyPermission "traceSettingsFile", "read,write";
permission java.lang.RuntimePermission "modifyThread";
permission java.net.SocketPermission "localhost:1024-", "listen";
permission java.net.SocketPermission "yourserver.yourcompany.com", connect";
```

 For more information on security relatingto user authentication and signed jars, read the official documentation forJava security architecture

Learn more about the WebSphere Applet client by running the Applet sample.You can install the Applet client sample from the *WebSphere Application Client* CD.This sample is called `HelloEJB` and is installed in the`product_installation`/`WSsamples/Client`subdirectory on the client machine.

4.7.2: J2EE application client programming model

The J2EE application client programming model provides the benefits of the J2EE platform for the Java client application. The J2EE platform offers the ability to seamlessly develop, assemble,deploy and launch a client application. The tooling provided with the WebSphere platform supports the seamless integration of these stages to help the developer create a client application from start to finish.

When a client application is developed using and adhering to the J2EE platform, the client application code is portable from one J2EE platform implementation to another. The client application package might require redeployment using each J2EE platform's deployment tool, but the code that comprises the client application will not change.

The J2EE application client runtime supplies a container that provides access to system services for the client application code. The client application code must contain a main method. The application client runtime invokes this main method after the environment is initialized and runs until the Java virtual machine is terminated.

The J2EE platform allows the J2EE application client to make use of "nicknames" or "short names," defined within the client application deployment descriptor. These deployment descriptors identify EJBs or local resources (JDBC, JMS, JavaMail and URL) for simplified resolution through the use of JNDI. This simplified resolution to the EJB reference and local resource reference also eliminates changes to the client application code when the underlying object or resource either changes or moves to a different server. Should these changes occur, the J2EE application client might require redeployment.

The J2EE application client also provides for initialization of the runtime environment for the client application. This initialization is unique for each client application and is defined by the deployment descriptor. In addition, the J2EE application client runtime provides support for security authentication to the EJBs and local resources.

The J2EE application client makes use of the RMI-IIOP protocol. The use of this protocol enables the client application to access EJB references and to make use of CORBA services that are provided by the J2EE platform implementation. Use of the RMI-IIOP protocol and the accessibility of CORBA services assist users in developing a client application that requires access to both EJB references and CORBA object references. When users combine the J2EE and CORBA environments or programming models in one client application, they need to understand the differences between the two programming models, and they must use and manage each appropriately.

The following table describes the advantages and disadvantages of the J2EE application client.

Advantages	Disadvantages
● Provides the user with the benefits of the J2EE platform. ● Allows the use of nicknames in the Deployment Descriptor for reference identity ● Client application code is portable across J2EE compliant platforms (may need to be redeployed for each distinct J2EE platform). ● Supports CORBA services (usage of CORBA services may render the client application code non-portable).	● A heavier client than the Java application thin client, and is not suited for downloading. ● Designed for use in an intranet environment. ● Requires distribution of the application to the client machine.

4.7.2.1: Resources referenced by a J2EE application client

J2EE application clients access resources by performing lookup operations in the JNDI name space. The application client runtime then provides a mapping of the resource names, used and configured by client application programs, to the actual resource objects.This allows client application programs to access different resources, such as test or productiondatabases, without the need for updates or recompiles.

To provide this service, the application client runtime requires information about the names and types ofresources used by the client application program.
The three types of resources a *J2EE application client* can reference are:

1. ***EJB references*** - are references to Enterprise Java Beans (EJBs) running on WebSphere Application Server.
2. ***Resource references*** - are references to other types of resources, such as:
    - JDBC databases
    - URLs
    - Java Message Service message queues
    - Java Mail
3. ***Environment entries*** - are references to simple data types that you would not want to code in your application program, such as timeout values or SQL query strings. The following Java basic data types are supported:

    `java.lang.Boolean` `java.lang.String` `java.lang.Integer` `java.lang.Double` `java.lang.Byte` `java.lang.Short` `java.lang.Long` `java.lang.Float`

The resource information is defined and configured using these two WebSphere Application Server tools:

- Application Assembly Tool (AAT) (used for the definition)
- Application Client Resource Configuration Tool (used for the configuration)

The ***Application Assembly Tool*** manages:

- EJB references
- non-client specific Resource references
- all Environment entries

The ***Application Client Resource Configuration Tool*** manages:

- client specific Resource references such as a JDBC Datasource name
  This information is stored with theclient application program in an Enterprise Archive File (`.ear` file).

4.7.2.2: Developing a J2EE application client

From an application developer's point of view, creating a *J2EE application client* program involves these steps:

1. Writing the client application program
2. Assembling the application client (using the Application Assembly Tool)
3. Assembling the Enterprise Archive (EAR) file
4. Distributing the EAR file
5. Configuring the application client resources
6. Launching the application client

---

1. Writing the client application program

Write the J2EE application client program on any development machine. At this stage, you do not require access to WebSphere Application Server.

A *J2EE application client* program is similar to a standard Java program in that it runs in its own Java virtual machine and is invoked at its main method. The J2EE application client program differs from a standardJava program because it uses the JNDI name space to access resources. In a standardJava program, the resource information is coded in the program.

Storing the resource information separately from the client application program makes the client application program portable and more flexible.

Using the `javax.naming.InitialContext` class, the client application program uses the lookup operation to access the JNDI name space.The `InitialContext` class provides the `lookup` method to locate resources. For more information on JNDI, see file, JNDI overview.

The following example illustrates how a client application program uses the `InitialContext` class:

```
import javax.naming.*public class myAppClient{    public static void main(String argv[])    {
InitialContext ctx = new InitialContext();        Object myObj =
ctx.lookup("java:comp/env/ejb/HelloBean");        HelloHome home
=(HelloHome)javax.rmi.PortableRemoteObject.narrow(myObj, HelloHome.class);
...    }}
```

In this example, the program is looking up an Enterprise Java Bean called *HelloBean*. The *HelloBean* EJB reference is located in the client JNDI name spaceat `java:comp/env/ejb/HelloBean`. Since the actual Enterprise Java Bean is running on the server, the application client runtimereturns a reference to ***HelloBean's*** home interface.

If the client application program's lookup was for aResource reference or an Environment entry, then lookup would return an instance of the configured typeas defined by the client application's Deployment Descriptor.For example, if the program's lookup was a JDBC datasource, the lookup would return an instance of`javax.sql.DataSource`.

2. Assembling the application client (using the Application Assembly Tool)

The JNDI name space knows what to return on a lookup because of the information that is assembled by the Application Assembly Tool (AAT).

Assemble the J2EE application client on any development machine that has the AATinstalled.

When you use the Application Assembly Tool to assemble your application client, you provide the *application client* runtime with the required information to initialize the execution environment for your client application program.Refer to the ***Application Assembly Tool*** description for implementation details.

Here is a list of things to keep in mind when you configure resources used by yourclient application program:

- When configuring Resource references and EJB references in the Application Assembly Tool, the **General** tab contains a required **Name** field. This field specifies where the application client runtime will bind the reference to the real object in the`java:comp/env` portion of the JNDI namespace. The application client runtimealways binds these references relative to `java:comp/env`. So, for the programmingexample above, you would specify `ejb/HelloBean` in the **Name** field on the**General** tab of the Application Assembly Tool, which would require the program to perform a lookup of `java:comp/env/ejb/HelloBean`. If the **Name** fieldwere set to `myString`, the resulting lookup would be `java:comp/env/myString`.
- When configuring Resource references in the Application Assembly Tool, the **Name** field on the **General** tab is used for:
  - binding a reference of that object type into the JNDI name space.
  - retrieving client specific resource configuration information that was configured using the Application Client Resource Configuration Tool.
- When configuring a Resource reference in the Application Assembly Tool, the value in the **Name** field on the **General** tab must match the value in the JNDI**Name** field on the **General** tab for the resource in theApplication Client Resource Configuration Tool.
- When configuring URL resources using the Client Resource Configuration Tool, the URL provider panel allows you to specify a protocol and a class that handles that protocol. If you want to use the default protocols, such as HTTP, you can leave those fields blank.
- When configuring Resource references using the Application Assembly Tool, the **General** tab contains a field called **Authorization**. This field can be set to either **Container** or **Application**. If the field is set to **Container**, then the application client runtime will use authorization information configured in the Application Client ResourceConfiguration tool for the resource. If the field is set to **Application**, thenthe Application Client runtime expects the user application to provide authorizationinformation for the resource. The application client runtime will ignore any authorization information configured with the Application Client Resource Configuration tool for that resource.

3. Assembling the Enterprise Archive (EAR) file

The application is contained in an Enterprise Archive or `.ear` file.The `.ear` file is composed of:

- EJB, Application Client, and user-defined modules or .jar files
- Web applications or `.war` files
- Metadata describing the applications or application `.xml` files

You must assemble the `.ear` file on the server machine.

4. Distributing the EAR file

The `.ear` file must be made accessible to those client machines that are configured to run this client.

If all the machines in your environment share the same image and platform, run the Application Client Resource Configuration Tool (ACRCT) on one machine to configure the external resources, and then distribute the configured `.ear` file to the other machines.

If your environment is set up with a variety of client installations and platforms, you must run the ACRCTfor each unique configuration.

The `.ear` files can either be distributed to the correct client machines, or made availableon a network drive.

See article Packaging and distributing Java clientsfor more information.

Distributing the `.ear` files is the responsibility of the system and network administrator.

5. Configuring the application client resources

If local resources are defined for use by the client application, run the ACRCT on the local machine to reconfigure the `.ear` file. Use the ACRCTto change the configuration. The ACRCT is the *Application Client Resource Configuration Tool* described in the previous steps. For example, the `.ear` file may contain a a DB2 resource, which is configured as `C:\DB2`. If, however, the user has DB2installed in directory, `D:\Program Files\DB2`, that user must use the ACRCT to create a local version of the `.ear` file.

6. Launching the application client

Using the fully assembled and configured `.ear` file,issue the `launchClient` command tolaunch the J2EE application client on the client machine.

> **Note:** Learn more about the WebSphere J2EE client by running the client sample.You can install the client sample from the *WebSphere Application Client* CD.On a server machine, the J2EE client sample is part of the samples gallery. See the "Application Client" section of `Samples.ear`.This sample is called `HelloEJB` and is installed in the`product_installation/WSsamples/Client`subdirectory on the client machine.

4.7.2.3: Troubleshooting guide for the J2EE application client

This section provides some debugging tips for resolving common J2EE application client problems. To use this troubleshooting guide, review the trace entries for one of the J2EE application client exceptions, and then locate the exception in the guide. See the Problem determination section formore information on starting traces and reading trace entries. Also see the WSCL Messages for a description ofapplication client messages. These messages help identify problems and can provide recovery information.

- java.lang.NoClassDefFoundError
- com.ibm.websphere.naming.CannotInstantiateObjectException
- javax.naming.ServiceUnavailableException
- javax.naming.CommunicationException
- javax.naming.NameNotFoundException
- java.lang.ClassCastException
- com.ibm.etools.archive.exception.OpenFailureException (message numbers WSCL0206E and WSCL0100E)

Error: `java.lang.NoClassDefFoundError`

Explanation: This exception is thrown when Java cannot load the specified class.

Possible causes:

- Invalid or non-existent class
- Classpath problem
- Manifest problem

User response: First check to determine if the specified class exists in a jar filewithin your ear file. If it does, make sure the path for the class is correct. For example, if you get the exception:

```
java.lang.NoClassDefFoundError: WebSphereSamples.HelloEJB.HelloHome
```

ensure the class `HelloHome` exists in one of the jar files in your ear file.If it exists, ensure the path for the class is `WebSphereSamples.HelloEJB`.

If both the class and path are correct, then it is a classpath issue. Most likely, you do not have the failing class's jar file specified in the client jar file's manifest. To check this, open your ear file with the Application Assembly Tool and click on the Application Client. Add the names of the other jar files in the ear file to the `Classpath` field. This exception is generally caused by a missing EJB module name from the `Classpath` field.

If you have multiple jars to enter in the `Classpath` field, be sure to separate the jar names with spaces.

If you still have the problem, you have a situation where a class is being loaded from the harddrive instead of the ear file. This is a very difficult situation to debug because the offending class is not the one specified in the exception. Instead, another class is loaded from the hard drivebefore the one specified in the exception. To correct this, review the classpathsspecified with the `-CCclasspath` option and the classpaths configured with theApplication Client Resource Configuration Tool. Look for classes that also exist in the ear file. You must resolve the situation whereone of the classes is found on the hard drive instead of in the `.ear` file. You do this by removingentries from the classpaths or by including the `.jar` files and classes in the `.ear` file instead of referencing them from the hard drive.

If you are using the `-CCclasspath` parameter or resourceclasspaths in the Application Client Resource Configuration Tool, and you have configuredmultiple jars or classes, verify they are separated with the correct character for your operating system. Unlike the classpath field in the Application Assembly Tool,these classpath fields use platform-specific separator characters, usually acolon (on UNIX platforms) or a semi-colon (on Windows).

Note: The system classpath is not used by the Application Client runtime if you use the `launchClient` batch or shell files. In this case, the system classpath wouldnot cause this problem. However, if you load the `launchClient` class directly, you dohave to search through the system classpath as well.

Return

Error: `com.ibm.websphere.naming.CannotInstantiateObjectException: Exception occurred while attempting to get an instance of the object for the specified reference object. [Root exception is javax.naming.NameNotFoundException: xxxxxxxxxx]`

Explanation: This exception occurs when you perform a lookup on an object that is not installed on the host server. Your program can look up the name in the local client JNDI name space, but received a `NameNotFoundException` exception because it is not located on the host server. One typical example is looking up an EJB that is not installed on the host server that you access.This exception might also occur if the JNDI name you configured in your ApplicationClient module does not match the actual JNDI name of the resource on the host server.

Possible causes:

- Incorrect host server invoked
- Resource is not defined
- Resource is not installed
- Application server is not started
- Invalid JNDI configuration

User response: If you are accessing the wrong host server, run the `launchClient` command again with the `-CCBootstrapHost` parameter specifying the correct host server name. If you are accessing the correct host server, use the WebSphere `dumpnamespace` command line tool tosee a listing of the host server's JNDI name space. If you do not see the failing object's name, the resourceis either not installed on the host server or the appropriate application server is not started.If you determine the resource is already installed and started, your JNDI name in your clientapplication does not match the global JNDI name on the host server. Use the Application Assembly Tool to compare the JNDI bindings value of the failing object's name in the client applicationto the JNDI bindings value of the object in the host server application. They must match.

Return

Error: `javax.naming.ServiceUnavailableException: Caught exception when resolving initial reference=NameService. Root exception is org.omg.CORBA.INTERNAL: JORB00105: In Profile.getIPAddress(), InetAddress.getByName( invalidhostname ) threw an UnknownHostException minor code: 0 completed: No`

Explanation: This exception occurs when you specify an invalid host server name.

Possible causes:

- Incorrect host server invoked
- Invalid host server name

User response: Run the `launchClient` command again and specify the correct name of your host server with the `-CCBootstrapHost` parameter.

Return

Error: `javax.naming.CommunicationException: Caught CORBA.COMM_FAILURE when resolving initial reference=WsnNameService. Root exception is org.omg.CORBA.COMM_FAILURE: minor code: 3 completed: No`

Explanation: This exception occurs when you run `launchClient` to a host server that does not have the Application Server started. You also receive this exceptionwhen you specify an invalid host server name. This might happen if you do not specify a host server name when you run `launchClient`. The default behavior is for `launchClient`to run to `localhost`, because WebSphere Application Server does not know the name of yourhost server. This default behavior only works when you are running the client on the samecomputer as where WebSphere Application Server is installed.

Possible causes:

- Incorrect host server invoked
- Invalid host server name
- Invalid reference to `localhost`
- Application server is not started
- Invalid bootstrap port

User Response: If you are not running to the correct host server, run the `launchClient` command again and specify the name of your host server with the `-CCBootstrapHost` parameter. Otherwise, start the Application Server on thehost server and run the `launchClient` command again.

Return

Error: `javax.naming.NameNotFoundException: Name comp/env/ejb not found in context "java:"`
Explanation: This exception is thrown when Java cannot locate the specified name in the localJNDI name space.

Possible causes:

- No binding information for the specified name
- Binding information for the specified name is incorrect
- Wrong class loader was used to load one of the program's classes

User Response: Open the ear file with the Application Assembly Tool and check the bindings for the failing name. Ensure this information is correct. If it is correct, you could have a class loader issue.

Return

Error: `java.lang.ClassCastException: Unable to loadclass: org.omg.stub.WebSphereSamples.HelloEJB._HelloHome_Stubat com.ibm.rmi.javax.rmi.PortableRemoteObject.narrow(portableRemoteObject.java:269)`
Explanation: This exception occurs when the application programattempts to narrow to the EJB's home class and the classloaders cannot find theEJB's client side bindings.

Note: The `HelloHome_Stub` reference in the **Error**description, is a sample

Possible causes:

- The files, `*_Stub.class` and `_Tie.class`, are notin the EJB .jar file

- Classloader could not find the classes

User Response: Look at the EJB `.jar` file located in the `.ear`and verify the class contains the EJB client side bindings. These are class files whosenames end in `_Stub` and `_Tie`. If these files are not present, then use the Application Assembly Toolto generate the binding classes. For more information, see article Generating deployment code for modules.If the binding classes are in the EJB `.jar` file, then you might have a classloader issue.

Return

Error: `WSCL0206E: File [EAR file name] is not avalid Enterprise Archive file. WSCL0100E: Exception received: com.ibm.etools.archive.exception.OpenFailureException`
Explanation: This error occurs when the Application Client runtime cannot read the Enterprise Archive file.

Possible cause: The most likely cause of this error is the EAR filecannot be found in the path specified on the `launchClient` command.

User Response: Verify that the path and filename specified on the `launchclient` command are correct. If you are running on Windows NTand the path and file name are correct, use a short version of the path and file name (8 character file name and3 character extension). For additional information, read the cause and recovery documentation for message WSCL0206E in the 8.2 Messages section.

Return

4.7.2.4: J2EE application client classloading overview

When you run your J2EE client application using the WebSphere Application ServerlaunchClient command, a hierarchy of classloadersis created to load classes used by your application. The parent classloader isused to load the WebSphere Application Client runtime and any classes placed in the WebSphere Application Client user directories. The directories used by this classloaderare defined by the WS_EXT_DIRS System property in the `product_installation/bin/setupcmdline` command shell.

As the J2EE Application Client runtime initializes, additional classloaders are createdas children of this parent classloader. If your client application uses resources such as JDBC, JMS, or URLs, adifferent classloader is created to load eachof those resources. Finally, a classloader is created to load classes within the `.ear` file. Before invoking your clientapplication's main method, this classloader is set as the thread's context classloader.

The system classpath is never used and is not part of thehierarchy of classloaders.

In order to package your client application correctly, you must understand which classloader loads your classes. When Java loads a class, the classloader used to loadthat class is assigned to it.Any classes subsequently loaded by that class will use that classloader or any ofits parents, but it will not use children classloaders.

Unfortunately, Java does not providea good way for determining which classloader loaded your classes. This makes itdifficult to debug classloading problems. See the Configuring the classpath fields section formore information on configuring the classpath fields in your application.

In some cases the WebSphere Application Client runtime can detect when your client application class is loaded by a different classloader from the one created for it by the WebSphere Application Client runtime. When that occurs, you will see message:
`WSCL0205W: The incorrect class loader was used to load {0}`.

This message occurs when your client application class is loaded by one of the parentclassloaders in the hierarchy. This is typically caused by having the sameclasses in the `.ear` file and on the hard drive. If one of the parentclassloaders locates a class, that classloader will load it before the Application Client runtimeclassloader. In some cases, your client application will still functioncorrectly. In most cases, however, you will receive "*class not found*"exceptions.

Configuring the classpath fields

When packaging your J2EE client application, you must configurevarious classpath fields. Ideally, you should package everything required by yourapplication into your `.ear` file. This is theeasiest way to distribute your J2EE client application to your clients. However, you should not package such resources as JDBC, JMS, or URLs. In the case of these resources, you want to use classpath referencesto access those classes on the hard drive. You might also have other classes installed on your client machines thatyou do not need to redistribute. In that case, you also want to useclasspath references to access the classes on the hard drive, as described below.

Referencing classes within the EAR file

WebSphere J2EE applications do not use the system path.Instead, use the MANIFEST Class-Path entries to refer to other classeswithin the `.ear` file. Configure these values using the module Classpath fieldsin the Application Assembly Tool. For example, if your client application needs to accessan Enterprise Java Bean, you would add the deployed EJB module's name to your applicationclient's Classpath field in the ApplicationAssembly Tool. The format of the Classpath field for each of thedifferent modules (Application Client, EJB, Web) is the same:

- The values must refer to `.jar` and `.class` files that are contained within the `.ear` file.

- The values must be relative to the root of the `.ear` file.
- The values cannot refer to absolute paths in the file systems.
- Multiple values must be separated by spaces, not colons or semi-colons.

  Note: This is Java's method for allowing applications to be platform-independent.

Typically, you add modules (`.jar` files) to the root of the `.ear` file. In this case, you only need to specify the name of the module (`.jar` file) in the Classpath field. If you choose to add a module with a path, you need tospecify the path relative to the root of the .ear file.

For referencing `.class` files, you must specify the directory relative tothe root of the `.ear` file. While the Application Assembly Tool allows you to add individual class files to the `.ear`file, it is recommended that these additional class files are packaged in a `.jar`file. That `.jar` file should then be added to the module Classpath fields. If you add `.class` files to the root of the `.ear` file, add "./" to the module Classpath fields.

Consider the following example directory structure in which the file myapp.ear contains an application client JAR file named client.jar and an EJB module called mybeans.jar. Additional classes reside in class1.jar and utility/class2.zip. A class named xyz.class is not packaged in a JAR file but is in the root of the EAR file.

Specify `"./ mybeans.jar utility/class2.zip class1.jar"` as the value of the Classpath property.

The search order is:

```
myapp.ear/client.jar
myapp.ear/mybeans.jar
myapp.ear/class1.jar
myapp.ear/utility/class2.zip
myapp.ear/xyz.class
```

View article the 6.4.1: Setting classpaths for more information.

Referencing classes that are not in the EAR file

You have two options to reference classes that are not contained in the `.ear` file. Which option you choose depends on the relationship of theexternal classes and the classes internal to the `.ear` file. You might use acombination of both options. Your options are:

1. Use the `product_installation/app` directory.

   Use this option when your external classes do ***not*** reference classes within the `.ear` file. One example would be stand-alone utility classes. To use this option, addyour `.jar` files to the `product_installation/app` directory.For `.class` files, add them to this directory in subdirectories that correspond to thepackage names.
2. If the external classes reference classes within the `.ear` file, thefirst option will not work because of the hierarchy ofWebSphere classloaders. In this case, you can do one of the following:
   - Package the external classes in the `.ear` file.
   - Use the `launchClient -CCclasspath` parameter.

     This parameter is specified at run-time and takes platform-specific classpath values, which means multiple values are separated by semi-colons or colons.

Refer to article 6.4.1 about installing application files into the environment, and setting classpaths, for a description of the WebSphere Application Server classloaders.There are many similarities between the client andthe server in this respect.

Resource classpaths

When you configure resources used by your client application using the Application Client Resource Configuration Tool, you can specify classpaths that are required by the resource. For example, if your application is using JDBC to a DB2 database, you want to add `db2java.zip` to the classpath field of the database provider. These classpath values are platform-specific and require semi-colons or colons to separate multiple values.

Using the launchClient API

If you use the launchClient shell/bat command, the WebSphere classloaderhierarchy is created for you. However, if you use the launchClient API, you mustperform this setup yourself. You should mimic thelaunchClient shell command in defining the Java system properties.

4.7.3: Java thin application client programming model

The Java application thin client provides the user a light weight, downloadable Java application runtime that is capable of interacting with Enterprise Java Beans. This client is designed to support those users who want a light weight Java client application programming environment without the overhead of the J2EE platform on the client machine. The programming model for this client is heavily influenced by the CORBA programming model but supports access to Enterprise Java Beans. When accessing Enterprise Java Beans from this client, the EJB object references can be considered CORBA object references by the client application.

There is no tooling support for this client for developing, assembling or deploying the client application. The user is responsible for developing the client application, generating the necessary client bindings for the EJB and CORBA objects, and bundling these pieces together to be installed on the client machine.

Client side bindings for Enterprise Java Beans are generated during the deployment phase of J2EE developmentusing the Application Assembly Tool. A Java application can utilize these bindings or you can generate client side bindingsusing the *rmic* command that is part of the IBM JDK installed withWebSphere Application Server. See article Packaging anddistributing Java clients for more information.

The Java application thin client provides the necessary runtime to support the communication needs between the client and the server.

The Java application thin client makes use of the RMI-IIOP protocol. The use of this protocol enables the client application to access not only EJB references and CORBA object references, but it also allows the client application to make use of any supported CORBA services. Use of the RMI-IIOP protocol and the accessibility of CORBA services can assist a user in developing a client application that needs to access both EJB references and CORBA object references. When users combine the J2EE and CORBA environments in one client application, they need to understand the differences between the two programming models, and they must use and manage each appropriately.

The Java application thin client runtime provides the necessary support for the client application for object resolution, security, RAS and other services. However, it does not support a container that provides ease of use to these services. For instance, there is no support for the use of "nicknames" for EJB or local resource resolution. When resolving to an EJB (using either JNDI or CosNaming) the client application must know the location of the name server and the fully qualified name that was used when the reference was bound into the namespace. When resolving to a local resource, the client application cannot resolve to the resource through a JNDI lookup. Instead the client application must explicitly create the connection to the resource using the appropriate API (JDBC, JMS, etc.). This client does not perform any initialization of any of the services that the client applicationmight require. For instance, the client application is responsible for the initialization of the naming service, either through CosNaming or JNDI.

The following table describes the advantages and disadvantages of the Java thin application client:

Advantages	Disadvantages
• A light-weight client suitable for download • Requires access to CosNaming or JNDI interfaces for EJB or CORBA object resolution	• Designed for use in an intranet environment • Lack of client runtime initialization of environment and services • Lack of built-in support for local resource resolution and configuration • Does not promote portability of client application code • Requires distribution of the application to the client machine.

4.7.3.1: Developing a Java application thin client

Install the Java application thin client from the *WebSphere Application Client* CDby selecting option "Java Application Thin Client" or "Java Application/Applet Thin Client."

The Java application thin client offers access to most of the client services that are available in the J2EE application client; however, these services are not as easily accessed in the thin client as they are in the J2EE application client. The J2EE client has the advantage of performing a simple JNDI namespace lookup to access the desired service or resource. The thin client must code explicitly for each resource in the client application. For example, looking up an EJB Home requires the following code in a J2EE application client:

```
        java.lang.Object ejbHome =  initialContext.lookup("java:/comp/env/ejb/MyEJBHome");
MyEJBHome = (MyEJBHome)javax.rmi.PortableRemoteObject.narrow(ejbHome,  MyEJBHome.class);
```

However, the code in a Java thin application client must be more explicit:

```
                        java.lang.Object ejbHome =
initialContext.lookup("the/fully/qualified/path/to/actual/home/in/namespace/MyEJBHome");
MyEJBHome = (MyEJBHome)javax.rmi.PortableRemoteObject.narrow(ejbHome,  MyEJBHome.class);
```

In this example, the J2EE application client accesses a logical name from the `java:/comp` namespace. The J2EE client runtime resolves that name to the physical location, and returns the reference to the client application. The thin client, on the other hand, must know the fully qualified physical location of the EJB Home in the namespace. If this location changes, the thin client application also must change the value placed on the lookup() statement. In the J2EE client, the client application is protected from these changes because it makes use of the logical name.A change might require a re-deploy of the EAR file, but the actual client application code remains the same.

The Java thin application client is a traditional Java application that contains a "main" function. WebSphere's Java thin application client provides runtime support for accessing remote EJBs, and provides the implementation for various services (security, WLM, and others). This client can also access CORBA objects and CORBA based services. When using both environments in one client application, the user is responsible for understanding the differences between the EJB and CORBA programming models and for managing both environments.

For instance, the CORBA programming model requires using the CORBA CosNaming name service for object resolution in a namespace while the EJB programming model requires using the JNDI name service. The client application must initialize and properly manage the use of these two naming services.

Another difference applies to the EJB model.The ORB is initialized using JNDI implementation in the EJB model, and the client application is unaware that an ORB is present. The CORBA model, however, requires the client application to explicitly initialize the ORB through the `ORB.init()` static method.

The Java application thin client provides a batch command that you can use to setthe CLASSPATH and JAVA_HOME environment variables to enable the Java application thinclient runtime.

Set the Java application thin client environment by using the ***setupClient***shell, located in:

```
product_installation\AppletClient\bin\setupClient.bat (on Windows)
product_installation/AppletClient/bin/setupClient.sh (on UNIX platforms)
```

After setting the environment variables, add your specific Java client applicationJAR files to the CLASSPATH and start your Java client application from this environment.

See article Packaging and distributing Java clientsfor more information.

4.7.3.2: Java thin application client code example

The code required by a Java application thin client to communicate with an enterprise java bean is similar to servlet code that communicates with enterprise java beans.

The following code example illustrates how a Java application thin client uses the `InitialContext` to do the following:

- Perform a lookup
- Narrow the returned object into the `EJBHome` object
- Invoke the `create` method.

Click a link to view the referenced line of code in the example. Each line in the code snippet is described in this next section.

1. The first three lines in the try section of the code example show how to:
    - Create a properties class
    - Set the initial context factory
    - Define the provider URL used during the lookup operation
2. The fields in the provider URL represent:

    `iiop://myComputer.myDomain.com:900`

iiop://	myComputer	myDomain.com	900
protocol	name of the server where WebSphere Application Server is installed	name of the domain for the server where WebSphere Application Server is installed	configured port Since port 900 is the default port value, thismay be omitted.

3. This line in the example shows how to:

    create an InitialContext class passing the Properties file
4. Now do a lookup the EJB Home on the server

    For more information on JNDI, see article 4.6.1: JNDI overview.
5. The `narrow` operation in this line:

    safely casts the object into an instance of `HelloHome`
6. Finally, call the create method on the HelloHome object to create a Hello object.

    You can also use `findByPrimary` key instead of `create`.Use the `findByPrimaryKey` method to find an existing Hello object.

Code example

```
import javax.naming.*;
import javax.rmi.*;
import java.rmi.*;
 import java.util.*;
 import javax.ejb.*;
import WebSphereSample.HelloEJB.*; //package for HelloEJB beans
     public class HelloClient
{
    public static void main(String argv[])
    {
     try
       {
            Properties props = new Properties();
            props.put(Context.INITIAL_CONTEXT_FACTORY,
        "com.ibm.websphere.naming.WsnInitialContextFactory");
            props.put(Context.PROVIDER_URL,
        "iiop://myComputer.myDomain.com:900");
            InitialContext ctx = new InitialContext(props);
            Object myObj = ctx.lookup("WSsamples/HelloEJBHome");
            HelloHome myHome = (HelloHome)
         javax.rmi.PortableRemoteObject.narrow(obj, HelloHome.class);
            Hello hello = myHome.create();
       }
      catch(NamingException e)
            ....
      catch(RemoteException e)
            ....
      catch(CreateException e)
            ....
     }
}
```

Learn more about the WebSphere Java application thin client by running the client sample.You can install the client sample from the *WebSphere Application Client* CD.This sample is called `HelloEJB` and is installed in the`product_installation`/`WSsamples/Client`subdirectory on the client machine.

4.7.4: Quick reference to Java client functions

Use the following table to identify the available functions in the different types of Java client.

Available functions	Applet client	J2EE Application client	Java thin application client
Provides all the benefits of a J2EE platform	No	Yes	No
Portable across all J2EE platforms	No	Yes	No
Provides the necessary runtime to support communication between client and server	Yes	Yes	Yes
Allows the use of nicknames in the deployment descriptors	No	Yes	No
Supports use of the RMI-IIOP protocol	Yes	Yes	Yes
Supports use of the HTTP protocol	Yes	No	No
Enables development of client applications that can access EJB references and CORBA object references	Yes	Yes	Yes
Enables the initialization of the client application's runtime environment	No	Yes	No
Supports security authentication to Enterprise Java Beans	No	Yes	Yes
Supports security authentication to local resources	No	Yes	No
Requires distribution of application to client machines	No	Yes	Yes

4.7.5: Quick reference to Java client topics

Use the following table to locate additional Java client topics.

Click a *Topics* entry to view a description.Click a *References* entry for additional information on that topic.

Topics	References
Java clients overview	• J2EE application model • J2EE architecture • Quick reference to Java client functions
Applet clients	• Developing an Applet client • Packaging, distributing, and installing • Tracing and logging
J2EE application clients	• Resources referenced by a J2EE application client • Developing a J2EE application client • Troubleshooting guide for a J2EE application client • J2EE application client client class loading overview • Packaging, distributing, and installing • Tracing and logging • Configuring application client resources • Launching Java application clients in the J2EE application client container • Assembling modules and setting properties for applications • Assembling J2EE application modules (.ear files) with theapplication assembly tool
Java thin application clients	• Developing a Java thin application client • Java thin application client code example • Packaging, distributing, and installing • Tracing and logging

4.7.6: Packaging and distributing Java client applications

After a client application has been developed the next step in the process is packaging and distributing the client application for use on client machines. Packaging consists of pulling together the various artifacts that the client application requires. Distributing applies to making the client application available on the target client machines. Each of the WebSphere Java clients differ slightly from each other in the packaging and distributing phases of the development process.These differences are described below.

Application client type	Packaging	Distribution
J2EE Application Client	Packaging of the WebSphere J2EE Application Client is accomplished through the Application Assembly Tool (AAT). This tool generates an Enterprise Archive (`.ear`) file as the output from the the assembly and deployment process. The `.ear` file contains all of the class files that are required by the client application to run. The `.ear` file must be deployed. The deployment can be done throughthe Application Assembly Tool or when the EJB modules within the `.ear`file are installed in WebSphere Application Server. The deployment phasegenerates the client bindings needed by the client application.	Distributing the J2EE Application Client `.ear` file to the target client machine that has WebSphere J2EE Application Client installed, is a manual process. WebSphere Application Server does not provide any tooling to assist in the distribution of the J2EE Application Client `.ear` files. The J2EE Application Client might require further configuration if the application makes use of external resources (such as: JDBC, JavaMail, JMS or URL). Perform this configuration with the Application Client Resource Configuration Tool. When the resource configuration is complete, the application can be started using the launchClient command.
Java Application Thin Client	The Java application thin client is packaged by manually collecting appropriate JAR files and Java classes to support the client application. The Enterprise Java Beans that the client application uses, require that client side bindings are available on the client target machine. The client side bindings are available from the deployed EJB JAR files. The `.jar` files, containing the Enterprise Java Beans, are invoked by the application. These JAR files are located in directory: `product_installation\InstalledApps\<YourEJBapplication.ear>\`	Distributing the Java application thin client JAR files to the target client machine where WebSphere Java Thin Application Client is installed, is a manual process. WebSphere Application Server does not provide any tooling to assist in the distribution of the JAR files. When the client application files are present on the target client machine, you must set up the environment. WebSphere Application Server provides a command that assists users in setting up the environment by defining several environment variables. Use the setupClient command located in the `product_installation\bin` directory. After running this command, add your client application JAR files to the CLASSPATH or use the `-classpath` parameter on the java command.

Applet Client	The Applet client is packaged manually by collecting the appropriate JAR files, Java classes, and HTML files to support the Applet. The Enterprise Java Beans that the Applet uses, require client side bindings. The client side bindings are available from the deployed EJB JAR files. What `.jar` files are used depends on the Enterprise Java Beans invoked. These JAR files are located inthe `product_installation\InstalledApps\<YourEJBapplication.ear>\` directory.	Distributing the Applet client to the target client machine that has the Applet client installed or to the target WebServer machine (if you want to make the Applet available for download), is a manual process. WebSphere Application Server does not provide any tooling to assist in the distribution of the JAR files.

4.7.7: Tracing and logging for the Java clients

Tracing and logging functions are available for the WebSphere client runtime. How this support is enabled and the level of support provided, differs for each client model.

- ## Applet client

  You enable the tracing and logging functions for ORB level tracing only, by specifying the following system properties in the Java ***Runtime parameters*** field of the WebSphere Application Server Java Plug-in Control Panel:

  ```
  -Dcom.ibm.CORBA.CommTrace=true
  -Dcom.ibm.CORBA.Debug=true
  ```

  All verbose, trace, and debug messages are sent to the Java console window on thebrowser. Applets restrict using files for trace output.

- ## J2EE Application Client

  You enable the tracing and logging functions by specifying one of the following flags on the launchClient command when starting the J2EE client application:

  - CCtrace
  - CCtracefile

  ### CCtrace flag

  The `-CCtrace` flag enables trace. You can trace all or specific components:

  - **`-CCtrace=true`**
    (This flag enables trace for all components and all events.)
  - **`-CCtrace=com.ibm.<component>=<entryexit | debug | event | all>=enabled`**
    (This flag enables trace for specific components. For example,
    -CCtrace=com.ibm.ws.client.*=all=enabled enables trace for all loggers with names starting with com.ibm.ws.client.)

    If the **-CCtrace** flag is not specified, trace is disabled.

  ### CCtracefile flag

  Use the `-CCtracefile` flag to send the trace output to a specific file:

  `-CCtracefile=<fully_qualified_output_filename>`

  (For example,
  -CCtracefile=c:\MyTraceFile.log directs the trace output to file, `c:\MyTraceFile.log`.)

  If the **-CCtracefile** flag is not set, all output is directed to stdout.

- ## Java thin application client

  You enable the tracing and logging functions by specifying the following system property on the java command when starting the client application:

  `-DtraceSettingsFile=<filename>`

  (Filename is the name of a properties file that must be placed in the classpath accessible by the application.)

  The properties file is used for specifying the output file and the components to enable for trace. When you install WebSphere Application Server, a sample trace settings properties file is provided in:
  `<product_installation>/properties/TraceSettings.properties`

  The `TraceSettings.properties` file looks like the following example:

```
# property to specify the fully qualified file name for the tracefile
traceFileName=c:\\MyTraceFile.log
# Specify trace strings here. Trace strings take the form of:
# logger={level}={type} where:
#     level = entryexit || debug || event || all
#     type =  enabled || disabled
# examples:
# com.ibm.ejs.ras.SharedLogBase=all=enabled enables all tracing for the single logger
#         created in class com.ibm.ejs.ras.SharedLogBase.
# com.ibm.ejs.*=debug=enabled enables debug tracing for all loggers with names starting
#         with com.ibm.ejs.
## Multiple trace strings can be specified, one per line.
com.ibm.ejs.ras.*=all=enabled
```

If you specify a filename but no trace string,only message events are written to the specified file. If you specify a filenameand a trace string, both message events and diagnostic trace entries are written to the specified file. If you do not specify a filename for the trace file, all output is directed to stdout.

6.6.24: Administering application client modules (overview)

Administration application client modules consists of the following.

Use the Application Assembly Tool to:

1. Creating the module
2. Setting deployment descriptor properties
3. Generating code for deployment

Use the Application Client Resource Configuration Tool to set additional configuration properties.

Classpath considerations

An important classpath-related setting to note is the Module Visibility. This application server setting impacts the portability of applications and standalone modules from other WebSphere Application Server versions and editions. If your existing module does not run as-is when you transfer it to Version 4.0, you might need to reassemble an existing module or change the module visibility setting.

See the information on setting classpaths for a full discussion of classpath considerations.

6.6.24.0: Application client module properties

These are set using the Application Assembly Tool. Refer to the "Assembly properties for client modules" property reference to set values for these properties.

6.6.24.0.a: Assembly properties for application client modules

File name (Required, String)

Specifies the file name of the application client module, relative to thetop level of the EAR file. If this is a stand-alone module, the filename is the full pathname of the archive.

Alternative DD

Specifies the file name for an alternative deployment descriptor file touse instead of the original deployment descriptor file in the module'sJAR file. This file is the postassembly version of the deploymentdescriptor file. (The original deployment descriptor file can be editedto resolve dependencies and security information. Directing the use ofthe alternative deployment descriptor allows you to keep the originaldeployment descriptor file intact). The value of the Alternative DDproperty must be the full path name of the deployment descriptor file relativeto the module's root directory. By convention, the file is in theALT-INF directory. If this property is not specified, the deploymentdescriptor file is read directly from the module's JAR file.

Classpath

Specifies the full class path containing the dependent code that is notcontained in the application client JAR file. Specify the valuesrelative to the root of the EAR file and separate the values withspaces. Absolute values that reference files or directories on the harddrive are ignored. To specify classes that are not in JAR files but arein the root of the EAR file, use a period and forward slash(./). Consider the following example directory structure inwhich the file myapp.ear contains an application client JAR file namedclient.jar. Additional classes reside in class1.jar andclass2.zip. A class named xyz.class is not packaged in aJAR file but is in the root of the EAR file.

```
myapp.ear/client.jarmyapp.ear/class1.jarmyapp.ear/class2.zipmyapp.ear/xyz.class
```

Specify `class1.jar class2.zip ./` as thevalue of the Classpath property. (Name only the directory for.class files.)

Display name (Required, String)

Specifies a short name that is intended to be displayed by GUIs.

Small icon

Specifies a JPEG or GIF file containing a small image (16x16pixels). The image is used as an icon to represent the applicationclient in a GUI.

Large icon

Specifies a JPEG or GIF file containing a large image (32x32pixels). The image is used as an icon to represent the applicationclient in a GUI.

Description

Contains text describing the application client.

Main class (Required, String)

Specifies the full path name of the main class for this applicationclient.

6.6.24.5: Administering application clients with Application Assembly Tool

An application client is a standalone Java program (in contrast to a Webbrowser-based program). An application client module is used toassemble the files that make up the application client into a singleunit. The Application Assembly Tool is used to create and edit modules,verify the archives, and generate deployment code. See the relatedtopics for links to concepts, instructions for creating an application clientmodule, and field help.

If the application client requires local resources, you must also use theApplication Client Resource Configuration Tool. This tool allows you todefine references to local resources other than enterprise beans (such as JDBCand JMS resources) on the machine where the application client resides.

6.6.24.5.1: Creating an application client

Application client modules can be created by using property dialog boxes orby using a wizard.

- Using the property dialog boxes
- Using the Create Application Client wizard

Using the property dialog boxes

Creating a new application client consists of specifying the files thatmake up the client and then adding assembly properties. To create a newapplication client:

1. Click **File** ->**New** -> **ApplicationClient**. The navigation pane displays a hierarchical structureused to build the contents of the module. The icons represent thecomponents, assembly properties, and files for the module. A propertydialog box containing general information about the application client isdisplayed in the property pane.
2. By default, the application client JAR file and the display name are thesame. It is recommended that you change the display name in theproperty pane.
3. By default, the temporary location of the application client JAR file isinstallation_directory`/bin`. You must specify a newfile name and location by clicking **File**->**Save**.
4. Enter the main class file name (required). Click **Browse**to locate the class file. By default, the root directory or archive isthe current archive. If needed, browse the file system for thedirectory or archive where class files reside. Click**Select**. The archive's file structure is displayed inthe window. Expand the structure and locate the files that youneed. Select the file and click **OK**.
5. Enter values for other properties as needed. View the help for 6.6.24.0.a: Assembly properties for application client modules. Click **OK**.
6. Define assembly properties for the application client.
    - Right-click the EJB References icon and click **New**. Aproperty dialog box for EJB References is displayed. View the help for 6.6.43.0.1: Assembly properties for EJB references. Enter values for each property and then click**OK**. Repeat to specify multiple EJB references. Thetop portion of the property pane lists each reference. Select thereference to view its corresponding property dialog box.
    - Right-click the Resource References and click **New**. Aproperty dialog box for Resource References is displayed. View the helpfor 6.6.43.0.2 Assembly properties for resource references. Enter values for each property and then click**OK**. Repeat to specify multiple resource references.
    - Right-click the Environment Entries and click **New.** Aproperty dialog box for Environment Entries is displayed. View the helpfor 6.6.34.0.a: Assembly properties for environment entries. Enter values for each property and then click**OK**. Repeat to define multiple environment variables.
7. Add files for the application client. In the navigation pane,right-click the Files icon and then choose **Add Files**. Usethe file browser to locate files to add. First, browse for the rootdirectory or archive where the files are located and click**Select**. If you are adding an entire archive, select thedirectory that contains the archive. The directory structure isdisplayed in the left pane. Browse the directory structure. Fromthe right pane, select one or more files to be added and click**Add**. If you select a directory and click **Add**, allfiles in the directory, including the directory, are added. Relativepath names are maintained. The selected files are displayed in theSelected Files window. Click **OK**. The file names,extensions, modification dates, sizes, and path names are displayed in theproperty pane.

8. Review the contents of the module and make any desired changes.
9. Click **File**->**Save** to save the module.

Using the Create Application Client wizard

Use this wizard to create an application client JAR file. Duringcreation, you specify the files that make up the application client.You also specify information such as references to other (external) enterprisebeans needed by the client.

Before you start the wizard, you must have the class files and other filesbelonging to the application client. When the wizard is completed, yourapplication client JAR file resides in the location that you specified.

To create an application client, click the **Wizards** icon on thetoolbar, and then click **Application Client**. Follow theinstructions on each panel.

- Specifying application client module properties
- Adding files
- Specifying additional application client module properties
- Choosing application client module icons
- Adding EJB references
- Adding resource references
- Adding environment entries
- Setting additional properties and saving the archive

Specifying application client module properties

On the **Specifying Application Client Module Properties**panel:

1. Indicate the application to which this module is to be added. If aparent application is not indicated, the module is created as a stand-aloneapplication.
2. Specify a display name for the application client (required). Thedisplay name is used to identify your application in the Application AssemblyTool and can be used by other tools.
3. Specify a file name for the application client (required). The filename specifies a location on your system for the JAR file to becreated.
4. Provide a short description of the application client (optional).
5. Click **Next**.

Adding files

On the **Adding Files** panel, specify the files that are to beassembled for the application client. To add or remove files:

1. Click **Add**. Use the file browser to locate files toadd. First, browse for the root directory or archive where the filesare located and click **Select**. If you are adding an entirearchive, select the directory that contains the archive. The directorystructure is displayed in the left pane. Browse the directorystructure. From the right pane, select one or more files to be addedand click **Add**. If you select a directory and click**Add**, all files in the directory, including the directory, areadded. Relative path names are maintained. The selected filesare displayed in the Selected Files window. Click **OK**.The files are displayed in a table on the wizard panel.

2. If you want to remove a file, select the file in the table and then click**Remove**.
3. Continue to add or remove files until you have the correct set offiles.
4. Click **Next**.

Specifying additional application client module properties

On the **Specifying Additional Application Client ModuleProperties** panel:

1. Specify the classpath and main class for the application client.View the help for 6.6.24.0.a: Assembly properties for application client modules.
2. Click **Next**.

Choosing application client module icons

On the **Choosing Application Client Module Icons** panel, specifyicons for your module.

1. Specify the full path name of a file containing a small icon, or click**Browse** to locate and select the file. The icon must be a GIFor JPEG image 16x16 pixels in size.
2. Specify a full path name of a file containing a large icon, or click**Browse** to locate and select the file. The icon must be a GIFor JPEG image 32x32 pixels in size.
3. Click **Next**.

Adding EJB references

On the **Adding EJB References** panel, specify the enterprise beansrequired by the application client.

1. Click **Add**. Enter the name of the enterprise bean to beused by the application client, the names of the bean's home and remoteinterfaces, and the bean type (required). View the help for 6.6.43.0.1: Assembly properties for EJB references. The EJB reference is displayed in a table on thewizard panel.
2. If the referenced bean is located in the module being created (or in theencompassing application), enter a name in the Link field (optional).If the bean is external to the module, leave the Link field blank. Youcan specify JNDI binding information later (by using the property dialog boxesor by using the administrative console).
3. To remove a reference, select the entry in the table and then click**Remove**.
4. Continue to add and remove references as needed.
5. Click **Next**.

Adding resource references

On the **Adding Resource References** panel, enter references toconnection factory objects for resource managers.

- Click **Add**. Enter values for the name, type, andauthorization mode of the resource reference, then click **OK**.View the help for 6.6.43.0.2 Assembly properties for resource references. The resource reference information is displayed in atable.
- To remove a reference, select the reference in the table and click**Remove**.
- Continue adding or removing references as needed.
- Click **Next**.

Adding environment entries

On the **Adding Environment Entries** panel, add environment entriesfor the application client.

- Click **Add**. Enter the name and type of the entry, thenclick **OK**. View the help for 6.6.34.0.a: Assembly properties for environment entries.
- To remove an environment entry, select the entry in the table and thenclick **Remove**.
- Continue adding or removing environment entries as needed.
- Click **Next**.

Setting additional properties and saving the archive

Click **Finish** to complete the wizard. To change settingsfor properties, click **Back** to return to the appropriatepanel. Make any needed changes, and then click**Finish**.

After you click **Finish**, the contents of the archive aredisplayed in the main window. You can continue adding or modifyingproperties as needed. For example, you can add bindinginformation. When you are finished editing the archive, click**File**->**Save**. Specify a name for the archive andclick **Save**.

6.6.0.9: Application Client Resource Configuration Tool for configuring client resources

The *Application Client Resource Configuration Tool* defines the resources for the client application. These configurations are stored in the client application .ear file, and are used by the WebSphere application client runtime for resolving and creating an instance of the resources for the client application.

Launching the tool

6.6.0.9.3: Removing objects from EAR files with the Application Client Resource Configuration Tool

During this task, you will remove (delete) an object from an EARfile for your application client. You can remove any particular J2EE resource or resource provider, including data sources and data source providers; URLs and URL providers;JMS providers, connection factories, and destinations; and JavaMail sessions. You cannot removethe default JavaMail provider, however.

1. Start the tool and open the EAR file from which you want to remove an object. The EAR file contents will be displayed in a tree view.

   Recommended. If you already had the EAR file open, and have made some changes, click **File** -> **Save** to save your work before preceding to delete an object. See below for a discussion.
2. In the tree, locate the object that you want to remove.
3. Do one of the following:
   - Click the object, then click **Edit** -> **Delete** from the tool menu bar.
   - Right-click the object to display its menu, then click **Delete**.
4. If you are sure that you want the deletion to take effect, click **File** -> **Save**.

 The option to delete an item does not offera confirmation dialog. As soon as you delete the item, it is gone. As a safeguard,consider saving your work right before you begin this task. That way, if you change yourmind after removing an item, you can close the EAR file without saving your changes, which willcancel your deletion. Be sure to do so immediately after the deletion, or you willalso lose any unsaved work that you performed since the deletion.

6.6.0.9a: Starting the Application Client Resource Configuration Tool and opening an EAR file

Start the graphical interface with the command:

```
clientConfig
```

To open an EAR file into the tool:

1. On the menu bar of the tool, click **File** -> **Open**.
2. Browse for the file that you want to open.
3. When you have found the file and selected it, click **Open**.

To save your changes to the file and close the tool:

1. On the menu bar of the tool, click **File** -> **Save**.
2. Click **File** -> **Exit**.

Now begin administering a resource type:

- JDBC providers and data sources
- JavaMail providers and sessions
- URL providers and URLs
- JMS providers, connection factories, and destinations

6.6.14.9: Administering data source providers and data sources with the Application Client Resource Configuration Tool

Use the Application Client Resource Configuration Tool to edit the configurations of data source providers (such as JDBC providers)and data sources, which are used by your application clients to access data fromdatabases.

Work with objects of this type by locating them in the tree that is displayed by the tool when you use it to openan EAR file. If file with which you are working contains data source providers and data sources, its tree will contain one or more of the following:

Resources -> ***application*.jar** -> **Data Source Providers** -> ***data_source_provider_instance***

where ***data_source_provider_instance*** isa particular data source provider.

If you expand the tree further, you will also see the **Data Sources** folders containing the data source instances for each data source provider instance.

6.6.14.9.1: Configuring new data source providers (JDBC drivers) with the Application Client Resource Configuration Tool

During this task, you will create new data source providers (also known as JDBC providers)for your client application. Note, in a separate administrative task, the Java code for the required data source provider must be installed on the client machine on which the client application resides.

To configure a new data source provider:

1. Start the tool and open the EAR file for which you want to configure the new data source provider. The EAR file contents will be displayed in a tree view.
2. From the tree, select the JAR file in which you wantto configure the new data source provider.
3. Expand the JAR file to view its contents.
4. Click the folder called **Data Source Providers**. Do one of the following:
    - Right-click the folder and select **New Provider**.
    - On the menu bar, click **Edit** -> **New**.
5. In the resulting property dialog, configure the data source provider properties.
6. When finished, click **OK**.
7. On the menu bar, click **File** -> **Save** to save your changes.

6.6.14.9.1.1: Configuring new data sources with the Application Client Resource Configuration Tool

During this task, you will create new data sources for your client application.

1. In the tree, click the data source provider for which you want to create a data source.
    - Configure a new data source provider.
    - Or, click an existing data source provider.
2. Expand the data source provider to view its **Data Sources** folder.
3. Click the folder. Do one of the following:
    - Right-click the folder and select **New Factory**.
    - On the menu bar, click **Edit** -> **New**.
4. In the resulting property dialog, configure the data source properties.
5. When finished, click **OK**.
6. On the menu bar, click **File** -> **Save** to save your changes.

6.6.14.9.3: Removing data source providers (JDBC drivers) and data sources with the Application Client Resource Configuration Tool

Please see "Removing objects from EAR files with theApplication Client Resource Configuration Tool", as this task is similar for all object types supported by the tool.

6.6.14.9.4: Updating data source and data source provider configurations with the Application Client Resource Configuration Tool

During this task, you will modify (update) the configuration of an existing data source ordata source provider.

1. Start the tool and open the EAR file containing the data source or data source provider. The EAR file contents will be displayed in a tree view.
2. From the tree, select the JAR file containing the data source or data source providerthat you want to update.
3. Expand the JAR file to view its contents.
4. Keep expanding the JAR file contents until you locate the particular data source or datasource provider that you want to update. When you find it, do one of the following:
    - Right-click the object and select **Properties**
    - On the menu bar, click **Edit** -> **Properties**
5. In the resulting property dialog, update the properties. For detailed field help, see:
    - Data source provider properties
    - Data source properties
6. When finished, click **OK**.
7. On the menu bar, click **File** -> **Save** to save your changes.

6.6.37.9: Administering JavaMail providers and sessions with the Application Client Resource Configuration Tool

Use the Application Client Resource Configuration Tool to edit the configurations of JavaMail sessions to be used by your application clients.

Work with objects of this type by locating them in the tree that is displayed by the tool when you use it to openan EAR file. If file with which you are working contains JavaMail sessions, its tree will contain one or more of the following:

Resources -> ***application*.jar** -> **JavaMail Providers** -> **Mail Provider** (a default mail provider) ->**JavaMail Sessions**

Inside the **JavaMail Sessions** folder will be JavaMail sessioninstances.

6.6.37.9.1: Configuring new JavaMail sessions with the Application Client Resource Configuration Tool

During this task, you will configure new mail sessions for your application client. The mail sessions will be associated with the preconfigured default mail provider supplied by the product.

To configure a new JavaMail Session:

1. Start the tool and open the EAR file for which you want to configure the new JavaMail session. The EAR file contents will be displayed in a tree view.
2. From the tree, select the JAR file in which you wantto configure the new JavaMail session.
3. Expand the JAR file to view its contents.
4. Click **JavaMail Providers** -> **MailProvider** -> **JavaMail Sessions**. Do one of the following:
    - Right-click the **JavaMail Sessions** folder and select **New Factory**.
    - On the menu bar, click **Edit** -> **New**.
5. In the resulting property dialog, configure the JavaMail session properties.
6. When finished, click **OK**.
7. On the menu bar, click **File** -> **Save** to save your changes.

6.6.37.9.3: Removing JavaMail sessions with the Application Client Resource Configuration Tool

Please see "Removing objects from EAR files with theApplication Client Resource Configuration Tool", as this task is similar for all object types supported by the tool.

6.6.37.9.4: Updating JavaMail session configurations with the Application Client Resource Configuration Tool

During this task, you will modify (update) the configuration of an existing JavaMail session. Note, you cannot update the default JavaMail provider, but you can viewits properties by performing similar steps.

1. Start the tool and open the EAR file containing the JavaMail session. The EAR file contents will be displayed in a tree view.
2. From the tree, select the JAR file containing the JavaMail session that you want to update.
3. Expand the JAR file to view its contents.
4. Keep expanding the JAR file contents until you locate the particular JavaMail session that you want to update. When you find it, do one of the following:
    - Right-click the object and select **Properties**
    - On the menu bar, click **Edit** -> **Properties**
5. In the resulting property dialog, update the properties. For detailed field help, see:
    - JavaMail session properties
6. When finished, click **OK**.
7. On the menu bar, click **File** -> **Save** to save your changes.

6.6.38.9: Administering URL providers and URLs with the Application Client Resource Configuration Tool

Use the Application Client Resource Configuration Tool to edit the configurations of URL providers and URLs to be used by your application clients.

Work with objects of this type by locating them in the tree that is displayed by the tool when you use it to openan EAR file. If file with which you are working contains URL providers and URLs, its tree will contain one or more of the following:

Resources -> ***application*.jar** -> **URL Providers** -> ***url_provider_instance***

where ***url_provider_instance*** isa particular URL provider.

If you expand the tree further, you will also see the **URLs** folders containing the URL instances for each URL provider instance.

6.6.38.9.1: Configuring new URL providers and URLs with the Application Client Resource Configuration Tool

During this task, you will create URL providers and URLs for your client application. Note, in a separate administrative task, the Java code for the required URL provider must be installed on the client machine on which the client application resides.

To configure a new URL provider:

1. Start the tool and open the EAR file for which you want to configure the new URL provider. The EAR file contents will be displayed in a tree view.
2. From the tree, select the JAR file in which you wantto configure the new URL provider.
3. Expand the JAR file to view its contents.
4. Click the folder called **URL Providers**. Do one of the following:
    - Right-click the folder and select **New Provider**.
    - On the menu bar, click **Edit** -> **New**.
5. In the resulting property dialog, configure the URL provider properties.
6. When finished, click **OK**.
7. On the menu bar, click **File** -> **Save** to save your changes.

6.6.38.9.1.1: Configuring new URLs with the Application Client Resource Configuration Tool

During this task, you will create URLs for your client application.

1. In the tree, click the URL provider for which you want to create a URL.
    - Configure a new URL provider.
    - Or, click an existing URL provider.
2. Expand the URL provider to view its **URLs** folder.
3. Click the folder. Do one of the following:
    - Right-click the folder and select **New Factory**.
    - On the menu bar, click **Edit** -> **New**.
4. In the resulting property dialog, configure the URL properties.
5. When finished, click **OK**.
6. On the menu bar, click **File** -> **Save** to save your changes.

6.6.38.9.3: Removing URL providers and URLs with the Application Client Resource Configuration Tool

Please see "Removing objects from EAR files with theApplication Client Resource Configuration Tool", as this task is similar for all object types supported by the tool.

6.6.38.9.4: Updating URL and URL provider configurations with the Application Client Resource Configuration Tool

During this task, you will modify (update) the configuration of an existing URL or URL provider.

1. Start the tool and open the EAR file containing the URL or URL provider. The EAR file contents will be displayed in a tree view.
2. From the tree, select the JAR file containing the URL or URL providerthat you want to update.
3. Expand the JAR file to view its contents.
4. Keep expanding the JAR file contents until you locate the particular URL or URL provider that you want to update. When you find it, do one of the following:
    - Right-click the object and select **Properties**
    - On the menu bar, click **Edit** -> **Properties**
5. In the resulting property dialog, update the properties. For detailed field help, see:
    - URL provider properties
    - URL properties
6. When finished, click **OK**.
7. On the menu bar, click **File** -> **Save** to save your changes.

6.6.39.9: Administering JMS providers, connection factories, and destinations with the Application Client Resource Configuration Tool

Use the Application Client Resource Configuration Tool to edit the configurations of JMS providers, JMS connectionfactories, and JMS destinations to be used by your application clients.

Work with objects of this type by locating them in the tree that is displayed by the tool when you use it to openan EAR file. If file with which you are working contains JMS providers, JMS connectionfactories, and JMS destinations, its tree will contain one or more of the following:

Resources -> ***application*.jar** -> **JMS Providers** -> ***jms_provider_instance***

where ***jms_provider_instance*** isa particular JMS provider.

If you expand the tree further, you will also see the **JMS Connection Factories** and **JMS Destinations** folders containing the connection factory and destination instances for each JMS provider instance.

6.6.39.9.1: Configuring new JMS providers with the Application Client Resource Configuration Tool

During this task, you will create new JMS provider configurations for your applicationclient. The client application can make use of a messaging service through the Java Message Service APIs. A JMS provider provides two kinds of J2EE factories. One is a JMS Connection factory, and the other is aJMS destination factory.

Note, in a separate administrative task, the JMS client must be installed on the client machine where the client application resides. The messaging product vendor must provide an implementation of the JMS client. For more information, see your messaging product documentation.

To configure a new JMS provider:

1. Start the tool and open the EAR file for which you want to configure the new JMS provider. The EAR file contents will be displayed in a tree view.
2. From the tree, select the JAR file in which you wantto configure the new JMS provider.
3. Expand the JAR file to view its contents.
4. Click the folder called **JMS Providers**. Do one of the following:
    - Right-click the folder and select **New Provider**.
    - On the menu bar, click **Edit** -> **New**.
5. In the resulting property dialog, configure the JMS provider properties.
6. When finished, click **OK**.
7. On the menu bar, click **File** -> **Save** to save your changes.

6.6.39.9.1.1: Configuring new connection factories with the Application Client Resource Configuration Tool

During this task, you will create a new JMS connection factory configuration for your applicationclient.

To configure a new connection factory:

1. In the tree, click the JMS provider for which you want to create a connection factory.
    - Configure a new JMS provider.
    - Or, click an existing JMS provider.
2. Expand the JMS provider to view its **JMS Connection Factories** folder.
3. Click the folder. Do one of the following:
    - Right-click the folder and select **New Factory**.
    - On the menu bar, click **Edit** -> **New**.
4. In the resulting property dialog, configure the JMS connection factory properties.
5. When finished, click **OK**.
6. On the menu bar, click **File** -> **Save** to save your changes.

6.6.39.9.1.2: Configuring new JMS destinations with the Application Client Resource Configuration Tool

During this task, you will create new JMS destination configuration for your applicationclient.

To configure a new destination:

1. In the tree, click the JMS provider for which you want to create a destination.
    - Configure a new JMS provider.
    - Or, click an existing JMS provider.
2. Expand the JMS provider to view its **JMS Destinations** folder.
3. Click the folder. Do one of the following:
    - Right-click the folder and select **New Factory**.
    - On the menu bar, click **Edit** -> **New**.
4. In the resulting property dialog, configure the JMS destination properties.
5. When finished, click **OK**.
6. On the menu bar, click **File** -> **Save** to save your changes.

6.6.39.9.3: Removing JMS providers, connection factories, and destinations with the Application Client Resource Configuration Tool

Please see "Removing objects from EAR files with theApplication Client Resource Configuration Tool", as this task is similar for all object types supported by the tool.

6.6.39.9.4: Updating JMS provider, connection factory, and destination configurations with the Application Client Resource Configuration Tool

During this task, you will modify (update) the configuration of an existing JMS provider, connection factory, or destination.

1. Start the tool and open the EAR file containing the JMS provider, connection factory, or destination. The EAR file contents will be displayed in a tree view.
2. From the tree, select the JAR file containing the JMS provider, connection factory, or destinationthat you want to update.
3. Expand the JAR file to view its contents.
4. Keep expanding the JAR file contents until you locate the particular JMS provider, connection factory, or destination that you want to update. When you find it, do one of the following:
    - Right-click the object and select **Properties**
    - On the menu bar, click **Edit** -> **Properties**
5. In the resulting property dialog, update the properties. For detailed field help, see:
    - JMS provider properties
    - JMS connection factory properties
    - JMS destination properties
6. When finished, click **OK**.
7. On the menu bar, click **File** -> **Save** to save your changes.