CHAPTER 11

Developing Java Clients

This chapter describes how to develop, assemble, and deploy Java clients in the following sections:

- "Introducing the Application Client Container" on page 17
- "Developing Clients Using the ACC" on page 19

Note – The Web Profile of the OracleGlassFish Server supports the EJB 3.1 Lite specification, which allows enterprise beans within web applications, among other features. The full GlassFish Server supports the entire EJB 3.1 specification. For details, see JSR 318 (`http://jcp.org/en/jsr/detail?id=318`).

Accordingly, the Application Client Container is supported only in the full GlassFish Server, not in the Web Profile.

JMS resources are supported only in the full GlassFish Server, not in the Web Profile. See Chapter 17, "Using the Java Message Service."

Introducing the Application Client Container

The Application Client Container (ACC) includes a set of Java classes, libraries, and other files that are required for and distributed with Java client programs that execute in their own Java Virtual Machine (JVM). The ACC manages the execution of Java EE application client components (application clients), which are used to access a variety of Java EE services (such as JMS resources, EJB components, web services, security, and so on.) from a JVM outside the Oracle GlassFish Server.

The ACC communicates with the GlassFish Server using RMI-IIOP protocol and manages the details of RMI-IIOP communication using the client ORB that is bundled with it. Compared to other Java EE containers, the ACC is lightweight.

For information about debugging application clients, see "Application Client Debugging" on page .

Note – Interoperability between application clients and GlassFish Servers running under different major versions is not supported.

ACC Security

The ACC determines when authentication is needed. This typically occurs when the client refers to an EJB component that requires authorization or when annotations in the client's `main` class trigger injection which, in turn, requires contact with the GlassFish Server's naming service. To authenticate the end user, the ACC prompts for any required information, such as a username and password. The ACC itself provides a very simple dialog box to prompt for and read these values.

The ACC integrates with the GlassFish Server's authentication system. It also supports SSL (Secure Socket Layer)/IIOP if configured and when necessary; see "Using RMI/IIOP Over SSL" on page 34.

You can provide an alternate implementation to gather authentication information, tailored to the needs of the application client. To do so, include the class to perform these duties in the application client and identify the fully-qualified name of this class in the `callback-handler` element of the `application-client.xml` descriptor for the client. The ACC uses this class instead of its default class for asking for and reading the authentication information. The class must implement the javax.security.auth.callback.CallbackHandler interface. See the Java EE specification, section 9.2, *Application Clients: Security*, for more details.

Application clients can use "Programmatic Login" on page .

ACC Naming

The client container enables the application clients to use the Java Naming and Directory Interface (JNDI) to look up Java EE services (such as JMS resources, EJB components, web services, security, and so on.) and to reference configurable parameters set at the time of deployment.

Application Client Annotation

Annotation is supported for the main class and the optional callback handler class in application clients. For more information, see "Deployment Descriptors and Annotations" in *GlassFish Server Open Source Edition 3.1 Application Deployment Guide*.

Java Web Start

Java Web Start allows your application client to be easily launched and automatically downloaded and updated. It is enabled for all application clients by default. For more information, see "Using Java Web Start" on page 22.

Application Client JAR File

In GlassFish Server 3.1, the downloaded appclient JAR file is smaller than in previous releases, with dependent classes in separate JAR files. When copying the downloaded appclient to another location, make sure to include the JAR files containing the dependent classes as well. You can also use the `asadmin` `get-client-stubs` command to retrieve the appclient and all associated application JAR files and place them in another location.

Developing Clients Using the ACC

This section describes the procedure to develop, assemble, and deploy client applications using the ACC. This section describes the following topics:

- "To Access an EJB Component From an Application Client" on page 19
- "To Access a JMS Resource From an Application Client" on page 21
- "Using Java Web Start" on page 22
- "Using the Embeddable ACC" on page 32
- "Running an Application Client Using the `appclient` Script" on page 33
- "Using the `package-appclient` Script" on page 33
- "The `client.policy` File" on page 34
- "Using RMI/IIOP Over SSL" on page 34
- "Connecting to a Remote EJB Module Through a Firewall" on page 35
- "Specifying a Splash Screen" on page 36
- "Setting Login Retries" on page 37
- "Using Libraries with Application Clients" on page 37

▼ To Access an EJB Component From an Application Client

1 **In your client code, reference the EJB component by using an `@EJB` annotation or by looking up the JNDI name as defined in the `ejb-jar.xml` file.**

For more information about naming and lookups, see "Accessing the Naming Context" on page .

If load balancing is enabled as in Step 7 and the EJB components being accessed are in a different cluster, the endpoint list must be included in the lookup, as follows:

`corbaname:`*host1*`:`*port1*`,`*host2*`:`*port2*`,`*...*`/NameService#ejb/`*jndi-name*

2 **Define the `@EJB` annotations or the `ejb-ref` elements in the `application-client.xml` file. Define the corresponding `ejb-ref` elements in the `glassfish-application-client.xml` file.**

For more information on the `glassfish-application-client.xml` file, see "The glassfish-application-client.xml file" in *GlassFish Server Open Source Edition 3.1 Application Deployment Guide*. For a general explanation of how to map JNDI names using reference elements, see "Mapping References" on page .

3 **Deploy the application client and EJB component together in an application.**

For more information on deployment, see the *GlassFish Server Open Source Edition 3.1 Application Deployment Guide*. To get the client JAR file, use the `--retrieve` option of the `asadmin deploy` command.

To retrieve the stubs and ties generated during deployment, use the `asadmin get-client-stubs` command. For details, see the *GlassFish Server Open Source Edition 3.1 Reference Manual*.

4 **Ensure that the client JAR file includes the following files:**

- A Java class to access the bean.
- `application-client.xml` - (optional) Java EE application client deployment descriptor.
- `glassfish-application-client.xml` - (optional) GlassFish Server specific client deployment descriptor. For information on the `glassfish-application-client.xml` file, see "The glassfish-application-client.xml file" in *GlassFish Server Open Source Edition 3.1 Application Deployment Guide*.
- The `MANIFEST.MF` file. This file contains a reference to the `main` class, which states the complete package prefix and class name of the Java client.

5 **Prepare the client machine.**

This step is not needed for Java Web Start. This step is not needed if the client and server machines are the same.

If you are using the `appclient` script, package the GlassFish Server system files required to launch application clients on remote systems using the `package-appclient` script, then retrieve the application client itself using the `asadmin get-client-stubs` command.

For more information, see "Using the `package-appclient` Script" on page 33 and the *GlassFish Server Open Source Edition 3.1 Reference Manual*.

6 **To access EJB components that are residing in a remote system, make the following changes to the `sun-acc.xml` file or the `appclient` script. This step is not needed for Java Web Start.**

- Define the `target-server` element's `address` and `port` attributes to reference the remote server machine and its ORB port. See "target-server" in *GlassFish Server Open Source Edition 3.1 Application Deployment Guide*.
- Use the `-targetserver` option of the appclient script to reference the remote server machine and its ORB port. For more information, see "Running an Application Client Using the `appclient` Script" on page 33.

To determine the ORB port on the remote server, use the `asadmin get` command. For example:

```
asadmin --host rmtsrv get server-config.iiop-service.iiop-listener.iiop-listener1.port
```

For more information about the `asadmin get` command, see the *GlassFish Server Open Source Edition 3.1 Reference Manual*.

7 **To set up load balancing and failover of remote EJB references, define at least two `target-server` elements in the `sun-acc.xml` file or the `appclient` script. This step is not needed for Java Web Start.**

If the GlassFish Server instance on which the application client is deployed participates in a cluster, the ACC finds all currently active IIOP endpoints in the cluster automatically. However, a client should have at least two endpoints specified for bootstrapping purposes, in case one of the endpoints has failed.

The `target-server` elements in the `sun-acc.xml` file specify one or more IIOP endpoints used for load balancing. The `address` attribute is an IPv4 address or host name, and the `port` attribute specifies the port number. See "client-container" in *GlassFish Server Open Source Edition 3.1 Application Deployment Guide*.

The `--targetserver` option of the appclient script specifies one or more IIOP endpoints used for load balancing. For more information, see "Running an Application Client Using the `appclient` Script" on page 33.

8 **Run the application client.**

See "Using Java Web Start" on page 22 or "Running an Application Client Using the `appclient` Script" on page 33.

▼ To Access a JMS Resource From an Application Client

1 **Create a JMS client.**

For detailed instructions on developing a JMS client, see "Chapter 33: The Java Message Service API" in the *The Java EE 6 Tutorial*.

2 **Next, configure a JMS resource on the GlassFish Server.**

For information on configuring JMS resources, see **Broken Link (Target ID: BEAOK)**.

3 Define the `@Resource` or `@Resources` annotations or the `resource-ref` elements in the `application-client.xml` file. Define the corresponding `resource-ref` elements in the `glassfish-application-client.xml` file.

For more information on the `glassfish-application-client.xml` file, see "The glassfish-application-client.xml file" in *GlassFish Server Open Source Edition 3.1 Application Deployment Guide*. For a general explanation of how to map JNDI names using reference elements, see "Mapping References" on page .

4 Ensure that the client JAR file includes the following files:

- A Java class to access the resource.
- `application-client.xml` - (optional) Java EE application client deployment descriptor.
- `glassfish-application-client.xml` - (optional) GlassFish Server specific client deployment descriptor. For information on the `glassfish-application-client.xml` file, see "The glassfish-application-client.xml file" in *GlassFish Server Open Source Edition 3.1 Application Deployment Guide*.
- The `MANIFEST.MF` file. This file contains a reference to the `main` class, which states the complete package prefix and class name of the Java client.

5 Prepare the client machine.

This step is not needed for Java Web Start. This step is not needed if the client and server machines are the same.

If you are using the `appclient` script, package the GlassFish Server system files required to launch application clients on remote systems using the `package-appclient` script, then retrieve the application client itself using the `asadmin` `get-client-stubs` command.

For more information, see "Using the `package-appclient` Script" on page 33 and the *GlassFish Server Open Source Edition 3.1 Reference Manual*.

6 Run the application client.

See "Using Java Web Start" on page 22 or "Running an Application Client Using the `appclient` Script" on page 33.

Using Java Web Start

Java Web Start allows your application client to be easily launched and automatically downloaded and updated. General information about Java Web Start is available at `http://java.sun.com/javase/technologies/desktop/javawebstart/index.jsp`.

Java Web Start is discussed in the following topics:

- "Enabling and Disabling Java Web Start" on page 23
- "Downloading and Launching an Application Client" on page 23

- “The Application Client URL” on page 24
- “Signing JAR Files Used in Java Web Start” on page 25
- “Error Handling” on page 27
- “Vendor Icon, Splash Screen, and Text” on page 27
- “Creating a Custom JNLP File” on page 27

Enabling and Disabling Java Web Start

Java Web Start is enabled for all application clients by default.

The application developer or deployer can specify that Java Web Start is always disabled for an application client by setting the value of the `eligible` element to `false` in the `glassfish-application-client.xml` file. See the *GlassFish Server Open Source Edition 3.1 Application Deployment Guide*.

The GlassFish Server administrator can disable Java Web Start for a previously deployed eligible application client using the `asadmin set` command.

To disable Java Web Start for all eligible application clients in an application, use the following command:

```
asadmin set applications.application.app-name.property.java-web-start-enabled="false"
```

To disable Java Web Start for a stand-alone eligible application client, use the following command:

```
asadmin set applications.application.module-name.property.java-web-start-enabled="false"
```

Setting `java-web-start-enabled="true"` re-enables Java Web Start for an eligible application client. For more information about the `asadmin set` command, see the *GlassFish Server Open Source Edition 3.1 Reference Manual*.

Downloading and Launching an Application Client

If Java Web Start is enabled for your deployed application client, you can launch it for testing. Simply click on the Launch button next to the application client or application's listing on the App Client Modules page in the Administration Console.

On other machines, you can download and launch the application client using Java Web Start in the following ways:

- Using a web browser, directly enter the URL for the application client. See “The Application Client URL” on page 24.
- Click on a link to the application client from a web page.
- Use the Java Web Start command `javaws`, specifying the URL of the application client as a command line argument.
- If the application has previously been downloaded using Java Web Start, you have additional alternatives.

- Use the desktop icon that Java Web Start created for the application client. When Java Web Start downloads an application client for the first time it asks you if such an icon should be created.
- Use the Java Web Start control panel to launch the application client.

When you launch an application client, Java Web Start contacts the server to see if a newer client version is available. This means you can redeploy an application client without having to worry about whether client machines have the latest version.

The Application Client URL

The default URL for an application or module generally is as follows:

`http://`*host*`:`*port*`/`*context-root*

The default URL for a stand-alone application client module is as follows:

`http://`*host*`:`*port*`/`*appclient-module-id*

The default URL for an application client module embedded within an application is as follows. Note that the relative path to the application client JAR file is included.

`http://`*host*`:`*port*`/`*application-id*`/`*appclient-path*

If the *context-root*, *appclient-module-id*, or *application-id* is not specified during deployment, the name of the JAR or EAR file without the extension is used. If the application client module or application is not in JAR or EAR file format, an *appclient-module-id* or *application-id* is generated.

Regardless of how the *context-root* or *id* is determined, it is written to the server log when you deploy the application. For details about naming, see "Naming Standards" in *GlassFish Server Open Source Edition 3.1 Application Deployment Guide*.

To set a different URL for an application client, use the `context-root` subelement of the `java-web-start-access` element in the `glassfish-application-client.xml` file. This overrides the *appclient-module-id* or *application-id*. See *GlassFish Server Open Source Edition 3.1 Application Deployment Guide*.

You can also pass arguments to the ACC or to the application client's `main` method as query parameters in the URL. If multiple application client arguments are specified, they are passed in the order specified.

A question mark separates the context root from the arguments. Ampersands (&) separate the arguments and their values. Each argument and each value must begin with `arg=`. Here is an example URL with a `-color` argument for a stand-alone application client. The `-color` argument is passed to the application client's `main` method.

```
http://localhost:8080/testClient?arg=-color&arg=red
```

Note – If you are using the `javaws` *URL* command to launch Java Web Start with a URL that contains arguments, enclose the URL in double quotes (") to avoid breaking the URL at the ampersand (&) symbol.

Ideally, you should build your production application clients with user-friendly interfaces that collect information which might otherwise be gathered as command-line arguments. This minimizes the degree to which users must customize the URLs that launch application clients using Java Web Start. Command-line argument support is useful in a development environment and for existing application clients that depend on it.

Signing JAR Files Used in Java Web Start

Java Web Start enforces a security sandbox. By default it grants any application, including application clients, only minimal privileges. Because Java Web Start applications can be so easily downloaded, Java Web Start provides protection from potentially harmful programs that might be accessible over the network. If an application requires a higher privilege level than the sandbox permits, the code that needs privileges must be in a JAR file that was signed.

When Java Web Start downloads such a signed JAR file, it displays information about the certificate that was used to sign the JAR if that certificate is not trusted. It then asks you whether you want to trust that signed code. If you agree, the code receives elevated permissions and runs. If you reject the signed code, Java Web Start does not start the downloaded application.

Your first Java Web Start launch of an application client is likely to involve this prompting because by default GlassFish Server uses a self-signed certificate that is not linked to a trusted authority.

The GlassFish Server serves two types of signed JAR files in response to Java Web Start requests. One type is a JAR file installed as part of the GlassFish Server, which starts an application client during a Java Web Start launch: *as-install*`/lib/gf-client.jar`.

The other type is a generated application client JAR file. As part of deployment, the GlassFish Server generates a new application client JAR file that contains classes, resources, and descriptors needed to run the application client on end-user systems. When you deploy an application with the `asadmin deploy` command's `--retrieve` option, use the `asadmin get-client-stubs` command, or select the Generate RMIStubs option from the EJB Modules deployment page in the Administration Console, this is one of the JAR files retrieved to your system. Because application clients need access beyond the minimal sandbox permissions to work in the Java Web Start environment, the generated application client JAR file must be signed before it can be downloaded to and executed on an end-user system.

A JAR file can be signed automatically or manually. The following sections describe the ways of signing JAR files.

- "Automatically Signing JAR Files" on page 26

- "Using the `jar-signing-alias` Deployment Property" on page 26

Automatically Signing JAR Files

The GlassFish Server automatically creates a signed version of the required JAR file if none exists. When a Java Web Start request for the `gf-client.jar` file arrives, the GlassFish Server looks for *domain-dir*`/java-web-start/gf-client.jar`. When a request for an application's generated application client JAR file arrives, the GlassFish Server looks in the directory *domain-dir*`/java-web-start/`*app-name* for a file with the same name as the generated JAR file created during deployment.

In either case, if the requested signed JAR file is absent or older than its unsigned counterpart, the GlassFish Server creates a signed version of the JAR file automatically and deposits it in the relevant directory. Whether the GlassFish Server just signed the JAR file or not, it serves the file from the *domain-dir*`/java-web-start` directory tree in response to the Java Web Start request.

To sign these JAR files, by default the GlassFish Server uses its self-signed certificate. When you create a new domain, either by installing the GlassFish Server or by using the `asadmin create-domain` command, the GlassFish Server creates a self-signed certificate and adds it to the domain's key store.

A self-signed certificate is generally untrustworthy because no certification authority vouches for its authenticity. The automatic signing feature uses the same certificate to create all required signed JAR files.

Using the jar-signing-alias Deployment Property

The `asadmin deploy` command property `jar-signing-alias` specifies the alias for the security certificate with which the application client container JAR file is signed.

Java Web Start won't execute code requiring elevated permissions unless it resides in a JAR file signed with a certificate that the user's system trusts. For your convenience, GlassFish Server signs the JAR file automatically using the self-signed certificate from the domain, `s1as`. Java Web Start then asks the user whether to trust the code and displays the GlassFish Server certificate information.

To sign this JAR file with a different certificate, first add the certificate to the domain keystore. You can use a certificate from a trusted authority, which avoids the Java Web Start prompt. To add a certificate to the domain keystore, see "Administering JSSE Certificates " in *Oracle GlassFish Server 3.0.1 Administration Guide*.

Next, deploy your application using the `jar-signing-alias` property. For example:

```
asadmin deploy --property jar-signing-alias=MyAlias MyApp.ear
```

For more information about the `asadmin deploy` command, see the *GlassFish Server Open Source Edition 3.1 Reference Manual*.

Error Handling

When an application client is launched using Java Web Start, any error that the application client logic does not catch and handle is written to `System.err` and displayed in a dialog box. This display appears if an error occurs even before the application client logic receives control. It also appears if the application client code does not catch and handle errors itself.

Vendor Icon, Splash Screen, and Text

To specify a vendor-specific icon, splash screen, text string, or a combination of these for Java Web Start download and launch screens, use the `vendor` element in the `glassfish-application-client.xml` file. The complete format of this element's data is as follows:

```
<vendor>icon-image-URI::splash-screen-image-URI::vendor-text</vendor>
```

The following example vendor element contains an icon, a splash screen, and a text string:

```
<vendor>images/icon.jpg::otherDir/splash.jpg::MyCorp, Inc.</vendor>
```

The following example vendor element contains an icon and a text string:

```
<vendor>images/icon.jpg::MyCorp, Inc.</vendor>
```

The following example vendor element contains a splash screen and a text string; note the initial double colon:

```
<vendor>::otherDir/splash.jpg::MyCorp, Inc.</vendor>
```

The following example vendor element contains only a text string:

```
<vendor>MyCorp, Inc.</vendor>
```

The default value is the text string `Application Client`.

For more information about the `glassfish-application-client.xml` file, see the *GlassFish Server Open Source Edition 3.1 Application Deployment Guide*.

You can also specify a vendor-specific icon, splash screen, text string, or a combination by using a custom JNLP file; see "Creating a Custom JNLP File" on page 27.

Creating a Custom JNLP File

You can partially customize the Java Network Launching Protocol (JNLP) file that GlassFish Server uses for Java Web Start.

Custom JNLP files are discussed in the following topics:

- "Specifying the JNLP File in the Deployment Descriptor" on page 28

- "Referring to JAR Files from the JNLP File" on page 28
- "Referring to Other JNLP Files" on page 29
- "Combining Custom and Automatically Generated Content" on page 29

For more information about JNLP, see the Java Web Start Architecture JNLP Specification and API Documentation (`http://java.sun.com/javase/technologies/desktop/javawebstart/download-spec.html`).

Specifying the JNLP File in the Deployment Descriptor

To specify a custom JNLP file for Java Web Start, use the `jnlp-doc` element in the `glassfish-application-client.xml` file. If none is specified, a default JNLP file is generated.

The value of the `jnlp-doc` element is a relative path with the following format:

`[`*path-to-JAR-in-EAR*`!]`*path-to-JNLP-in-JAR*

The default *path-to-JAR-in-EAR* is the current application client JAR file. For example, if the JNLP file is in the application client JAR file at `custom/myInfo.jnlp`, the element value would look like this:

```
<java-web-start-access>
   <jnlp-doc>custom/myInfo.jnlp</jnlp-doc>
</java-web-start-access>
```

If the application client is inside an EAR file, you can place the custom JNLP file inside another JAR file in the EAR. For example, if the JNLP file is in a JAR file at `other/myLib.jar`, the element value would look like this, with an exclamation point (`!`) separating the path to the JAR from the path in the JAR:

```
<java-web-start-access>
   <jnlp-doc>other/myLib.jar!custom/myInfo.jnlp</jnlp-doc>
</java-web-start-access>
```

For more information about the `glassfish-application-client.xml` file, see the *Oracle GlassFish Server 3.0.1 Application Deployment Guide*.

Referring to JAR Files from the JNLP File

As with any JNLP document, the custom JNLP file can refer to JAR files the application client requires.

Do not specify every JAR on which the client depends. GlassFish Server automatically handles JAR files that the Java EE specification requires to be available to the application client. This includes JAR files listed in the application client JAR file's manifest `Class-Path` and JAR files in the EAR file's library directory (if any) and their transitive closures. The custom JNLP file should specify only those JAR files the client needs that GlassFish Server would not otherwise include.

Package these JAR files in the EAR file, as with any JAR file required by an application client. Use relative URIs in the `<jar href="...">` and `<nativelib href="...">` elements to point to the JAR files. The codebase that GlassFish Server assigns for the final client JNLP file corresponds to the top level of the EAR file. Therefore, relative `href` references correspond directly to the relative path to the JAR files within the EAR file.

Neither the Java EE specification nor GlassFish Server supports packaging JAR files inside the application client JAR file itself. Nothing prevents this, but GlassFish Server does no special processing of such JAR files. They do not appear in the runtime class path and they cannot be referenced from the custom JNLP file.

Referring to Other JNLP Files

The JNLP file can also refer to other custom JNLP files using `<extension href="..."/>` elements. To be consistent with relative `href` references to JAR files, the relative `href` references to JNLP files are resolved within the EAR file. You can place these JNLP files directly in the EAR file or inside JAR files that the EAR file contains. Use one of these formats for these `href` references:

`[`*path-to-JAR-in-EAR*`!]`*path-to-JNLP-in-JAR*

path-to-JNLP-in-EAR

Note that these formats are *not* equivalent to the format of the `jnlp-doc` element in the `glassfish-application-client.xml` file.

These formats follow the standard entry-within-a-JAR URI syntax and semantics. Support for this syntax comes from the automated Java Web Start support in GlassFish Server. This is not a feature of Java Web Start or the JNLP standard.

Combining Custom and Automatically Generated Content

GlassFish Server recognizes these types of content in the JNLP file:

- Owned — GlassFish Server owns the content and ignores any custom content
- Merged — Automatically generated content and custom content are merged
- Defaulted — Custom content is used if present, otherwise default content is provided

You can compose a complete JNLP file and package it with the application client. GlassFish Server then combines it with its automatically generated JNLP file. You can also provide content that only adds to or replaces what GlassFish Server generates. The custom content must conform to the general structure of the JNLP format so that GlassFish Server can properly place it in the final JNLP file.

For example, to specify a single native library to be included only for Windows systems, the new element to add might be as follows:

```
<nativelib href="windows/myLib.jar"/>
```

However, you must indicate where in the overall document this element belongs. The actual custom JNLP file should look like this:

```
<jnlp>
   <resources os="Windows">
      <nativelib href="windows/myLib.jar"/>
   </resources>
</jnlp>
```

GlassFish Server provides default `<information>` and `<resources>` elements, without specifying attributes such as `os`, `arch`, `platform`, or `locale`. GlassFish Server merges its own content within those elements with custom content under those elements. Further, you can provide your own `<information>` and `<resources>` elements (and fragments within them) that specify at least one of these attributes.

In general, you can perform the following customizations:

- Override the GlassFish Server defaults for the child elements of `<information>` elements that have no attribute settings for `os`, `arch`, `platform`, and `locale`. Among these child elements are `<title>`, `<vendor>`, `<description>`, `<icon>`, and so on.
- Add `<information>` elements with `os`, `arch`, `platform`, or `locale` settings. You can also add child elements.
- Add child elements of `<resources>` elements that have no attribute settings for `os`, `arch`, or `locale`. Among these child elements are `<jar>`, `<property>`, `<nativelib>`, and so on. You can also customize attributes of the `<java>` child element.
- Add `<resources>` elements that specify at least one of `os`, `arch`, or `locale`. You can also add child elements.

This flexibility allows you to add JAR files to the application (including platform-specific native libraries) and set properties to control the behavior of your application clients.

The following tables provide more detail about what parts of the JNLP file you can add to and modify.

TABLE 11–1 Owned JNLP File Content

JNLP File Fragment	Description
`<jnlp codebase="`*xxx*`" ...>`	GlassFish Server controls this content for application clients packaged in EAR files. The developer controls this content for application clients packaged in WAR files.
`<jnlp href="`*xxx*`" ...>`	GlassFish Server controls this content for application clients packaged in EAR files. The developer controls this content for application clients packaged in WAR files.

TABLE 11–1 Owned JNLP File Content *(Continued)*

JNLP File Fragment	Description

`<jnlp>`
`  <security>`
	GlassFish Server must control the permissions requested for each JNLP file. All permissions are needed for the main file, which launches the ACC. The permissions requested for other JNLP documents depend on whether the JAR files referenced in those documents are signed.

`<jnlp>`
`  <application-desc>`
`    <argument> ...`
	GlassFish Server sets the `main-class` and the arguments to be passed to the client.

TABLE 11–2 Defaulted JNLP File Content

JNLP File Fragment	Description
`<jnlp spec="`*xxx*`" ...>`	Specifies the JNLP specification version.

`<jnlp>`
`  <information [no-attributes]>`
	Specifies the application title, vendor, home page, various description text values, icon images, and whether offline execution is allowed.

`<jnlp>`
`  <resources [no-attributes]>`
`    <java version="`*xxx*`"`
`        java-vm-args="`*yyy*`" ...>`
	Specifies the Java SE version or selected VM parameter settings.

TABLE 11–3 Merged JNLP File Content

JNLP File Fragment	Description

`<jnlp>`
`  <information [attributes]>`
	You can specify one or more of the `os`, `arch`, `platform`, and `locale` attributes for the `<information>` element. You can also specify child elements; GlassFish Server provides no default children.

`<jnlp>`
`  <resources [attributes]>`
	You can specify one or more of the `os`, `arch`, `platform`, and `locale` attributes for the `<resources>` element. You can also specify child elements; GlassFish Server provides no default children.

`<jnlp>`
`  <resources [no-attributes]>`
`    <jar ...>`
	Adds JAR files to be included in the application to the JAR files provided by GlassFish Server.

`<jnlp>`
`  <resources [no-attributes]>`
`    <nativelib ...>`
	Adds native libraries to be included in the application. Each entry in a JAR listed in a `<nativelib>` element must be a native library for the correct platform. The full syntax of the `<nativelib>` element lets the developer specify the platform for that native library.

TABLE 11–3 Merged JNLP File Content (Continued)

JNLP File Fragment	Description

`<jnlp>`
`  <resources [no-attributes]>`
`    <property ...>`
	Adds system properties to be included in the application to the system properties defined by GlassFish Server.

`<jnlp>`
`  <resources [no-attributes]>`
`    <extension ...>`
	Specifies another custom JNLP file.

`<jnlp>`
`  <component-desc ...>`
	Includes another custom JNLP file that specifies a component extension.

`<jnlp>`
`  <installer-desc ...>`
	Includes another custom JNLP file that specifies an installer extension.

Using the Embeddable ACC

You can embed the ACC into your application client. If you place the *as-install*`/lib/gf-client.jar` file in your runtime classpath, your application creates the ACC after your application code has started, then requests that the ACC start the application client portion. The basic model for coding is as follows:

1. Create a builder object.
2. Operate on the builder to configure the ACC.
3. Obtain a new ACC instance from the builder.
4. Present a client archive or class to the ACC instance.
5. Start the client running within the newly created ACC instance.

Your code should follow this general pattern:

```
// one TargetServer for each ORB endpoint for bootstrapping
TargetServer[] servers = ...;

// Get a builder to set up the ACC
AppClientContainer.Builder builder = AppClientContainer.newBuilder(servers);

// Fine-tune the ACC's configuration.  Note ability to "chain" invocations.
builder.callbackHandler("com.acme.MyHandler").authRealm("myRealm"); // Modify config

// Get a container for a client.
URI clientURI = ...; // URI to the client JAR
AppClientContainer acc = builder.newContainer(clientURI);

or

Class mainClass = ...;
AppClientContainer acc = builder.newContainer(mainClass);

// In either case, start the client running.
String[] appArgs = ...;
```

```
acc.startClient(appArgs); // Start the client

...

acc.close(); // close the ACC(optional)
```

The ACC loads the application client's `main` class, performs any required injection, and transfers control to the `static` `main` method. The ACC's `run` method returns to the calling application as soon as the client's `main` method returns to the ACC.

If the application client's `main` method starts any asynchronous activity, that work continues after the ACC returns. The ACC has no knowledge of whether the client's `main` method triggers asynchronous work. Therefore, if the client causes work on threads other than the calling thread, and if the embedding application needs to know when the client's asynchronous work completes, the embedding application and the client must agree on how this happens.

The ACC's shutdown handling is invoked from the ACC's `close` method. The calling application can invoke `acc.close()` to close down any services started by the ACC. If the application client code started any asynchronous activity that might still depend on ACC services, invoking `close` before that asynchronous activity completes could cause unpredictable and undesirable results. The shutdown handling is also run automatically at VM shutdown if the code has not invoked `close` before then.

The ACC does not prevent the calling application from creating or running more than one ACC instance during a single execution of the application either serially or concurrently. However, other services used by the ACC (transaction manager, security, ORB, and so on) might or might not support such serial or concurrent reuse.

Running an Application Client Using the `appclient` Script

To run an application client, you can launch the ACC using the `appclient` script, whether or not Java Web Start is enabled. This is optional. This script is located in the *as-install*`/bin` directory. For details, see the *GlassFish Server Open Source Edition 3.1 Reference Manual*.

Using the `package-appclient` Script

You can package the GlassFish Server system files required to launch application clients on remote systems into a single JAR file using the `package-appclient` script. This is optional. This script is located in the *as-install*`/bin` directory. For details, see the *GlassFish Server Open Source Edition 3.1 Reference Manual*.

The `client.policy` File

The `client.policy` file is the J2SE policy file used by the application client. Each application client has a `client.policy` file. The default policy file limits the permissions of Java EE deployed application clients to the minimal set of permissions required for these applications to operate correctly. If an application client requires more than this default set of permissions, edit the `client.policy` file to add the custom permissions that your application client needs. Use the J2SE standard policy tool or any text editor to edit this file.

For more information on using the J2SE policy tool, see `http://java.sun.com/docs/books/tutorial/security/tour2/index.html`.

For more information about the permissions you can set in the `client.policy` file, see `http://java.sun.com/javase/6/docs/technotes/guides/security/permissions.html`.

Using RMI/IIOP Over SSL

You can configure RMI/IIOP over SSL in two ways: using a username and password, or using a client certificate.

To use a username and password, configure the `ior-security-config` element in the `glassfish-ejb-jar.xml` file. The following configuration establishes SSL between an application client and an EJB component using a username and password. The user has to login to the ACC using either the `sun-acc.xml` mechanism or the "Programmatic Login" on page mechanism.

```
<ior-security-config>
  <transport-config>
    <integrity>required</integrity>
    <confidentiality>required</confidentiality>
    <establish-trust-in-target>supported</establish-trust-in-target>
    <establish-trust-in-client>none</establish-trust-in-client>
  </transport-config>
  <as-context>
    <auth-method>username_password</auth-method>
    <realm>default</realm>
    <required>true</required>
  </as-context>
 <sas-context>
    <caller-propagation>none</caller-propagation>
 </sas-context>
</ior-security-config>
```

For more information about the `glassfish-ejb-jar.xml` and `sun-acc.xml` files, see the *GlassFish Server Open Source Edition 3.1 Application Deployment Guide*.

To use a client certificate, configure the `ior-security-config` element in the `glassfish-ejb-jar.xml` file. The following configuration establishes SSL between an application client and an EJB component using a client certificate.

```
<ior-security-config>
  <transport-config>
    <integrity>required</integrity>
    <confidentiality>required</confidentiality>
    <establish-trust-in-target>supported</establish-trust-in-target>
    <establish-trust-in-client>required</establish-trust-in-client>
  </transport-config>
  <as-context>
    <auth-method>none</auth-method>
    <realm>default</realm>
    <required>false</required>
  </as-context>
  <sas-context>
    <caller-propagation>none</caller-propagation>
  </sas-context>
</ior-security-config>
```

To use a client certificate, you must also specify the system properties for the keystore and truststore to be used in establishing SSL. To use SSL with the Application Client Container (ACC), you need to set these system properties in one of the following ways:

- Use the new syntax of the `appclient` script and specify the system properties as JVM options. See "Running an Application Client Using the `appclient` Script" on page 33.
- Set the environment variable `VMARGS` in the shell. For example, in the `ksh` or `bash` shell, the command to set this environment variable would be as follows:

  ```
  export VMARGS="-Djavax.net.ssl.keyStore=${keystore.db.file}
  -Djavax.net.ssl.trustStore=${truststore.db.file}
  -Djavax.net.ssl.keyStorePass word=${ssl.password}
  -Djavax.net.ssl.trustStorePassword=${ssl.password}"
  ```
- Optionally, you can set the `env` element using Ant. For example:

  ```
  <target name="runclient">
    <exec executable="${S1AS_HOME}/bin/appclient">
      <env key="VMARGS" value=" -Djavax.net.ssl.keyStore=${keystore.db.file}
        -Djavax.net.ssl.trustStore=${truststore.db.file}
        -Djavax.net.ssl.keyStorePasword=${ssl.password}
        -Djavax.net.ssl.trustStorePassword=${ssl.password}"/>
      <arg value="-client"/>
      <arg value="${appClient.jar}"/>
    </exec>
  </target>
  ```

Connecting to a Remote EJB Module Through a Firewall

To deploy and run an application client that connects to an EJB module on a GlassFish Server instance that is behind a firewall, you must set ORB Virtual Address Agent Implementation (ORBVAA) options. Use the `asadmin` `create-jvm-options` command as follows:

```
asadmin create-jvm-options -Dcom.sun.corba.ee.ORBVAAHost=public-IP-adress
asadmin create-jvm-options -Dcom.sun.corba.ee.ORBVAAPort=public-port
```

```
asadmin create-jvm-options
-Dcom.sun.corba.ee.ORBUserConfigurators.com.sun.corba.ee.impl.plugin.hwlb.VirtualAddressAgentImpl=x
```

Set the `ORBVAAHost` and `ORBVAAPort` options to the host and port of the public address. The `ORBUserConfigurators` option tells the ORB to create an instance of the `VirtualAddressAgentImpl` class and invoke the `configure` method on the resulting object, which must implement the com.sun.corba.ee.spi.orb.ORBConfigurator interface. The `ORBUserConfigurators` value doesn't matter. Together, these options create an ORB that in turn creates `Object` references (the underlying implementation of remote EJB references) containing the public address, while the ORB listens on the private address specified for the IIOP port in the GlassFish Server configuration.

Specifying a Splash Screen

Java SE 6 offers splash screen support, either through a Java command-line option or a manifest entry in the application's JAR file. To take advantage of this Java SE feature in your application client, you can do one of the following:

- Create the appclient JAR file so that its manifest contains a `SplashScreen-Image` entry that specifies the path to the image in the client. The `java` command displays the splash screen before starting the ACC or your client, just as with any Java application.
- Use the new `appclient ... -jar` launch format, using the `-splash` command-line option at runtime or the `SplashScreen-Image` manifest entry at development time. See "Running an Application Client Using the `appclient` Script" on page 33.
- In the environment that runs the `appclient` script, set the `VMOPTS` environment variable to include the `-splash` option before invoking the `appclient` script to launch the client.
- Build an application client that uses the embeddable ACC feature and specify the splash screen image using one of the following:
  - The `-splash` option of the `java` command
  - `SplashScreen-Image` in the manifest for your program (not the manifest for the application client)

  See "Using the Embeddable ACC" on page 32.

During application (EAR file) deployment, the GlassFish Server generates façade JAR files, one for the application and one for each application client in the application. During application client module deployment, the GlassFish Server generates a single facade JAR for the application client. The `appclient` script supports splash screens inside the application client JAR only if you launch an application client facade or appclient client JAR. If you launch the facade for an application or the undeployed application itself, the `appclient` script cannot take advantage of the Java SE 6 splash screen feature.

Setting Login Retries

You can set a JVM option using the `appclient` script that determines the number of login retries allowed. This option is `-Dorg.glassfish.appclient.acc.maxLoginRetries=`*n* where *n* is a positive integer. The default number of retries is 3.

This retry loop happens when the ACC attempts to perform injection if you annotated the client's `main` class (for example, using `@Resource`). If instead of annotations your client uses the `InitialContext` explicitly to look up remote resources, the retry loop does not apply. In this case, you could write logic to catch an exception around the lookup and retry explicitly.

For details about the `appclient` script syntax, see the *GlassFish Server Open Source Edition 3.1 Reference Manual*.

Using Libraries with Application Clients

The Libraries field in the Administration Console's deployment page and the `--libraries` option of the `asadmin deploy` command do not apply to application clients. Neither do the *as-install*`/lib`, *domain-dir*`/lib`, and *domain-dir*`/lib/classes` directories comprising the Common Class Loader. These apply only to applications and modules deployed to the server. For more information, see Chapter 2, "Class Loaders."

To use libraries with an application client, package the application client in an application (EAR file). Then, either place the libraries in the `/lib` directory of the EAR file or specify their location in the application client JAR file's manifest `Class-Path`.