CHAPTER 18

Administering the Java Naming and Directory Interface (JNDI) Service

The Java Naming and Directory Interface (JNDI) API is used for accessing different kinds of naming and directory services. Java EE components locate objects by invoking the JNDI lookup method.

The following topics are addressed here:

- "About JNDI" on page 21
- "Administering JNDI Resources" on page 23

Instructions for accomplishing the tasks in this chapter by using the Administration Console are contained in the Administration Console online help.

About JNDI

By making calls to the JNDI API, applications locate resources and other program objects. A *resource* is a program object that provides connections to systems, such as database servers and messaging systems. A JDBC resource is sometimes referred to as a data source. Each resource object is identified by a unique, people-friendly name, called the *JNDI name*. A resource object and its JNDI name are bound together by the naming and directory service, which is included with the GlassFish Server.

When a new name-object binding is entered into the JNDI, a new resource is created.

The following topics are addressed here:

- "Java EE Naming Environment" on page 22
- "How the Naming Environment and the Container Work Together" on page 22
- "Naming References and Binding Information" on page 23

Java EE Naming Environment

JNDI names are bound to their objects by the naming and directory service that is provided by a Java EE server. Because Java EE components access this service through the JNDI API, the object usually uses its JNDI name. For example, the JNDI name of the PointBase database is `jdbc/Pointbase`. At startup, the GlassFish Server reads information from the configuration file and automatically adds JNDI database names to the name space, one of which is `jdbc/Pointbase`.

Java EE application clients, enterprise beans, and web components must have access to a JNDI naming environment.

The application component's naming environment is the mechanism that allows customization of the application component's business logic during deployment or assembly. This environment allows you to customize the application component without needing to access or change the source code off the component. A Java EE container implements the provides the environment to the application component instance as a *JNDI naming context*.

How the Naming Environment and the Container Work Together

The application component's environment is used as follows:

- The application component's business methods access the environment using the JNDI interfaces. In the deployment descriptor, the application component provider declares all the environment entries that the application component expects to be provided in its environment at runtime.
- The container provides an implementation of the JNDI naming context that stores the application component environment. The container also provides the tools that allow the deployer to create and manage the environment of each application component.
- A deployer uses the tools provided by the container to initialize the environment entries that are declared in the application component's deployment descriptor. The deployer sets and modifies the values of the environment entries.
- The container makes the JNDI context available to the application component instances at runtime. These instances use the JNDI interfaces to obtain the values of the environment entries.

Each application component defines its own set of environment entries. All instances of an application component within the same container share the same environment entries. Application component instances are not allowed to modify the environment at runtime.

Naming References and Binding Information

A *resource reference* is an element in a deployment descriptor that identifies the component's coded name for the resource. For example, `jdbc/SavingsAccountDB`. More specifically, the coded name references a connection factory for the resource.

The JNDI name of a resource and the resource reference name are not the same. This approach to naming requires that you map the two names before deployment, but it also decouples components from resources. Because of this decoupling, if at a later time the component needs to access a different resource, the name does not need to change. This flexibility makes it easier for you to assemble Java EE applications from preexisting components.

The following table lists JNDI lookups and their associated resource references for the Java EE resources used by the GlassFish Server.

TABLE 18–1 JNDI Lookup Names and Their Associated References

JNDI Lookup Name	Associated Resource Reference
`java:comp/env`	Application environment entries
`java:comp/env/jdbc`	JDBC DataSource resource manager connection factories
`java:comp/env/ejb`	EJB References
`java:comp/UserTransaction`	UserTransaction references
`java:comp/env/mail`	JavaMail Session Connection Factories
`java:comp/env/url`	URL Connection Factories
`java:comp/env/jms`	JMS Connection Factories and Destinations
`java:comp/ORB`	ORB instance shared across application components

Administering JNDI Resources

Within GlassFish Server, you can configure your environment for custom and external JNDI resources. A custom resource accesses a local JNDI repository; an external resource accesses an external JNDI repository. Both types of resources need user-specified factory class elements, JNDI name attributes, and so on.

- "Administering Custom JNDI Resources" on page 24
- "Administering External JNDI Resources" on page 26

Administering Custom JNDI Resources

A custom resource specifies a custom server-wide resource object factory that implements the `javax.naming.spi.ObjectFactory` interface.

The following topics are addressed here:

- "To Create a Custom JNDI Resource" on page 24
- "To List Custom JNDI Resources" on page 24
- "To Update a Custom JNDI Resource" on page 25
- "To Delete a Custom JNDI Resource" on page 25

▼ To Create a Custom JNDI Resource

Use the `create-custom-resource` subcommand in remote mode to create a custom resource.

1. **Ensure that the server is running.**

   Remote subcommands require a running server.

2. **Create a custom resource by using the `create-custom-resource(1)` subcommand.**

   Information on properties for the subcommand is contained in this help page.

3. **Restart GlassFish Server.**

   See "To Restart a Domain" on page .

Example 18–1 Creating a Custom Resource

This example creates a custom resource named `sample-custom-resource`.

```
asadmin> create-custom-resource --restype topic --factoryclass com.imq.topic
sample_custom_resource
Command create-custom-resource executed successfully.
```

See Also You can also view the full syntax and options of the subcommand by typing `asadmin help create-custom-resource` at the command line.

▼ To List Custom JNDI Resources

Use the `list-custom-resources` subcommand in remote mode to list the existing custom resources.

1. **Ensure that the server is running.**

   Remote subcommands require a running server.

2. **List the custom resources by using the `list-custom-resources(1)` subcommand.**

Example 18–2 Listing Custom Resources

This example lists the existing custom resources.

```
asadmin> list-custom-resources
sample_custom_resource01
sample_custom_resource02
Command list-custom-resources executed successfully
```

See Also You can also view the full syntax and options of the subcommand by typing `asadmin help list-custom-resources` at the command line.

▼ To Update a Custom JNDI Resource

1. **List the custom resources by using the `list-custom-resources(1)` subcommand.**
2. **Use the `set(1)` subcommand to modify a custom JNDI resource.**

Example 18–3 Updating a Custom JNDI Resource

This example modifies a custom resource.

```
asadmin> set server.resources.custom-resource.custom
/my-custom-resource.property.value=2010server.resources.custom-resource.custom
/my-custom-resource.property.value=2010
```

▼ To Delete a Custom JNDI Resource

Use the `delete-custom-resource` subcommand in remote mode to delete a custom resource.

1. **Ensure that the server is running.**

   Remote subcommands require a running server.

2. **List the custom resources by using the `list-custom-resources(1)` subcommand.**
3. **Delete a custom resource by using the `delete-custom-resource(1)` subcommand.**

Example 18–4 Deleting a Custom Resource

This example deletes a custom resource named `sample-custom-resource`.

```
asadmin> delete-custom-resource sample_custom_resource
Command delete-custom-resource executed successfully.
```

See Also You can also view the full syntax and options of the subcommand by typing `asadmin help delete-custom-resource` at the command line.

Administering External JNDI Resources

Applications running on GlassFish Server often require access to resources stored in an external JNDI repository. For example, generic Java objects might be stored in an LDAP server according to the Java schema. External JNDI resource elements let you configure such external resource repositories.

The following topics are addressed here:

- "To Register an External JNDI Resource" on page 26
- "To List External JNDI Resources" on page 27
- "To List External JNDI Entries" on page 27
- "To Update an External JNDI Resource" on page 28
- "To Delete an External JNDI Resource" on page 28
- "Example of Using an External JNDI Resource" on page 28
- "To Disable GlassFish Server V2 Vendor-Specific JNDI Names" on page 29

▼ To Register an External JNDI Resource

Use the `create-jndi-resource` subcommand in remote mode to register an external JNDI resource.

Before You Begin The external JNDI factory must implement the `javax.naming.spi.InitialContextFactory` interface.

1 **Ensure that the server is running.**

Remote subcommands require a running server.

2 **Register an external JNDI resource by using the `create-jndi-resource(1)` subcommand.**

Information on properties for the subcommand is contained in this help page.

3 **Restart GlassFish Server.**

See "To Restart a Domain" on page .

Example 18–5 Registering an External JNDI Resource

In This example `sample_jndi_resource` is registered.

```
asadmin> create-jndi-resource --jndilookupname sample_jndi
--restype queue --factoryclass sampleClass --description "this is a sample jndi
resource" sample_jndi_resource
Command create-jndi-resource executed successfully
```

See Also You can also view the full syntax and options of the subcommand by typing `asadmin help create-jndi-resource` at the command line.

▼ To List External JNDI Resources

Use the `list-jndi-resources` subcommand in remote mode to list all existing JNDI resources.

1. **Ensure that the server is running.**

   Remote subcommands require a running server.

2. **List the existing JNDI resources by using the `list-jndi-resources(1)` subcommand.**

Example 18–6 Listing JNDI Resources

This example lists the JNDI resources.

```
asadmin> list-jndi-resources
jndi_resource1
jndi_resource2
jndi_resource3
Command list-jndi-resources executed successfully
```

See Also You can also view the full syntax and options of the subcommand by typing `asadmin help list-jndi-resources` at the command line.

▼ To List External JNDI Entries

Use the `list-jndi-entries` subcommand in remote mode to browse and list the entries in the JNDI tree. You can either list all entries, or you can specify the JNDI context or subcontext to list specific entries.

1. **Ensure that the server is running.**

   Remote subcommands require a running server.

2. **List the JNDI entries for a configuration by using the `list-jndi-entries(1)` subcommand.**

Example 18–7 Listing JNDI Entries

This example lists all the JNDI entries for the naming service.

```
asadmin> list-jndi-entries
jndi_entry03
jndi_entry72
jndi_entry76
Command list-jndi-resources executed successfully
```

See Also You can also view the full syntax and options of the subcommand by typing `asadmin help list-jndi-entries` at the command line.

▼ To Update an External JNDI Resource

1 **List the existing JNDI resources by using the`list-jndi-resources(1)` subcommand.**

2 **Use the `set(1)` subcommand to modify an external JNDI resource.**

Example 18–8 Updating an External JNDI Resource

This example modifies an external resource.

```
asadmin> set server.resources.external-jndi-resource.my-jndi-resource.
jndi-lookup-name=bar server.resources.external-jndi-resource.my-jndi-resource.jndi-lookup-name=bar
```

▼ To Delete an External JNDI Resource

Use the `delete-jndi-resource` subcommand in remote mode to remove a JNDI resource.

1 **Ensure that the server is running.**

Remote subcommands require a running server.

2 **Remove an external JNDI entry by using the `delete-jndi-resource(1)` subcommand.**

Example 18–9 Deleting an External JNDI Resource

This example deletes an external JNDI resource:

```
asadmin> delete-jndi-resource jndi_resource2
Command delete-jndi-resource executed successfully.
```

See Also You can also view the full syntax and options of the subcommand by typing `asadmin help delete-jndi-resource` at the command line.

Example of Using an External JNDI Resource

```
<resources>
 <!-- external-jndi-resource element specifies how to access Java EE resources
 -- stored in an external JNDI repository. This example
 -- illustrates how to access a java object stored in LDAP.
 -- factory-class element specifies the JNDI InitialContext factory that
 -- needs to be used to access the resource factory. property element
 -- corresponds to the environment applicable to the external JNDI context
 -- and jndi-lookup-name refers to the JNDI name to lookup to fetch the
 -- designated (in this case the java) object.
 -->
  <external-jndi-resource jndi-name="test/myBean"
      jndi-lookup-name="cn=myBean"
      res-type="test.myBean"
      factory-class="com.sun.jndi.ldap.LdapCtxFactory">
    <property name="PROVIDER-URL" value="ldap://ldapserver:389/o=myObjects" />
    <property name="SECURITY_AUTHENTICATION" value="simple" />
    <property name="SECURITY_PRINCIPAL", value="cn=joeSmith, o=Engineering" />
    <property name="SECURITY_CREDENTIALS" value="changeit" />
  </external-jndi-resource>
</resources>
```

▼ To Disable GlassFish Server V2 Vendor-Specific JNDI Names

The EJB 3.1 specification supported by GlassFish Server 3.1 defines portable EJB JNDI names. Because of this, there is less need to continue to use older vendor-specific JNDI names.

By default, GlassFish Server V2–specific JNDI names are applied automatically by GlassFish Server 3.1 for backward compatibility. However, this can lead to some ease-of-use issues. For example, deploying two different applications containing a Remote EJB component that exposes the same remote interface causes a conflict between the default JNDI names.

The default handling of V2–specific JNDI names in GlassFish Server 3.1 can be managed with the `asadmin` command or with the `disable-nonportable-jndi-names` boolean property for the `ejb-container` element in `glassfish-ejb-jar.xml`.

● **Use the `asadmin` command or directly modify the `glassfish-ejb-jar.xml` file to set the `disable-nonportable-jndi-names` property.**

- **Using the `asadmin` command:**

  ```
  asadmin> set server.ejb-container.property.disable-nonportable-jndi-names="true"
  ```

- **Directly modifying the `glassfish-ejb-jar.xml` file.**

  a. **Add The `disable-nonportable-jndi-names` property to the `ejb-container` element in `glassfish-ejb-jar.xml`.**

  b. **Set the value of the `disable-nonportable-jndi-names` boolean, as desired.**

     - `false` — Enables the automatic use of GlassFish Server V2–specific JNDI names. This is the default setting.
     - `true` — Disables the automatic use of V2–specific JNDI names. In all cases, 3.1-compatible JNDI names will be used.

  c. **Save the `glassfish-ejb-jar.xml` file and restart the GlassFish Server domain.**

     This setting applies to all EJBs deployed to the server.