

Utilizing the LDAP Extensions and Controls for JNDI in a Directory Environment

Welcome to the Novell *Utilizing the LDAP Extensions and Controls for JNDI in a Directory Environment* course. This course is intended to provide you with the information you need to utilize Novell's JNDI LDAP Extensions in your own programming projects.

Contents:

- Course Purpose
- Introduction to JNDI
- Setting up Your Work Environment
- Directory Concepts
- LDAP
- eDirectory Overview
- Overview of JNDI
- JNDI Naming Explored
- JNDI a History
- The Doc-u-matic, a JNDI Application
- Resources
- Conclusion

Course Description	This course will help you understand what JNDI is, what it does for the Java Programming Language, and how to use it to make your programming life easier.
Objective	You will be able to understand the JNDI Interface and how it makes interacting with Java Naming and Directories much simpler.
Estimated time to complete this course	It will take about 6-8 hours to complete this course.

Prerequisites	<ul style="list-style-type: none"> • Entry level Java Programming skills • A basic understanding of directories (classes, objects, attributes, syntaxes, values, etc.) • The ability to build Java applications with a command line development environment (Java from Sun's JDK.) • Familiarity with ConsoleOne or NWAdmin used to administer eDirectory. • A Novell DeveloperNet subscription (the Electronic Level subscription is free!)
Required Items	Computer workstation capable of running the JDK v2.x.
Optional Items	None
Required Setup	The most recent version of an Internet Browser (Netscape or Internet Explorer will work fine).
Development Environment	<ul style="list-style-type: none"> • JDK 2.x • JNDI Provider from Sun Microsystems • LDAP Provider and Directory such as Novell's eDirectory • JAVAC Java Compiler from Sun or another compatible Java compiler

Course Purpose

The purpose of this course is to help you understand JNDI, what it does for the Java Programming Language, and how to use it. This course covers:

- Introduction to JNDI
- Setting up Your Work Environment
- Directory Concepts
- LDAP
- eDirectory Overview
- Overview of JNDI
- JNDI Naming Explored
- JNDI a History
- The Doc-u-matic, a JNDI Application
- Resources
- Conclusion

Introduction to JNDI

JNDI (Java Naming and Directory Interface) is an industry-wide, open interface that gives developers a common interface for navigating the many naming systems that exist in the computing world today. JNDI greatly simplifies the code needed to browse directory services such as eDirectory, X.500, and LDAP.

The LDAP Extensions and Controls for JNDI are LDAP v3-compliant and include support for the virtual list views and server-side sorting controls available on eDirectory. Because it uses LDAP, it has no dependencies on the Novell NetWare Client software.

This course covers LDAP and JNDI along with Novell's JNDI LDAP Extensions.

Setting up Your Work Environment

This section contains information for using the LDAP Extensions and Controls for JNDI. First, let's look at dependencies that are required to make all of this work:

- JDK 1.17b or higher. We recommend JDK 1.3.1 due to a problem that is corrected in this build.
- A JSSE-compliant security provider to create SSL connections. For more information about SSL, see the DeveloperNet University *SSL Security Course*.
- NDS eDirectory v8.5 or greater to use the extensions for naming context and replica management.

Configuration

Configuration can utilize the JDK 1.3 or JDK 1.2 (JDK 1.3 is recommended).

JDK 1.3

1. Set your `JAVA_HOME` to your JDK directory. For example, set `JAVA_HOME=C:\jdk1.3`.
2. Include Novell's `novbp.jar`, which is included in the Novell JNDI download, in your `CLASSPATH`.

JDK 1.2

1. Set your `JAVA_HOME` to your JDK directory. For example, set `JAVA_HOME=C:\jdk1.2.2`.
2. Include Sun's `jndi.jar`, `ldap.jar`, and `providerutil.jar` in your `CLASSPATH`.
3. Include Novell's `novbp.jar`, which is included in the Novell JNDI download, in your `CLASSPATH`.

JDK 1.1.7b

1. Include JDK1.1.7b classes.zip in your CLASSPATH.
2. Include Sun's jndi.jar, ldap.jar, and providerutil.jar in your CLASSPATH.
3. Include Novell's novbp.jar, which is included in the Novell JDNI download, in your CLASSPATH.

Note: When attempting one of the following extended operations using JVM version 1.3 or lower, a null pointer exception will be thrown from LdapCtx.java. This issue will be resolved in JDK1.3.1. It is recommended that you use JDK1.3.1 once it becomes available.

AbortNamingContextOperation
AddReplica
ChangeReplicaType
CreateNamingContext
CreateOrphanNamingContext
MergeNamingContexts
NamingContextSync
ReceiveAllUpdates
RefreshLDAPServer
RemoveOrphanNamingContext
RemoveReplica
SchemaSync
SendAllUpdates
SetReplicationFilter
TriggerBackgroundProcess

Directory Concepts

Directory Services have been around for several years, but mostly in the enterprise environment. A Directory Service allows you to easily manage your network resources. Simply put, a Directory is a database of objects that represent network resources, such as network users, servers, printers, print queues, and applications. A Directory can be described as a hierarchal tree. Typically a Directory is stored as a set of database files on a server. A NetWare server stores these files on volume SYS. If no file system volumes are present, the server stores the Directory database files in an installation subdirectory. Typically a Directory can be replicated on multiple servers.

Directory Trees

Directories use trees to keep track of information. A Directory tree can be thought of as a distributed database that contains directory information about objects. That is, the database contains objects and the attributes, or properties, that describe those objects. Each Directory tree maintains its own database of objects. Because it is a distributed database, the database is usually contained on more than one server.

The information in a Directory tree does not describe the physical layout of the network. It usually describes the logical organization of the business. The Directory tree is usually organized into subtrees that reflect the different departments and units in an organization. In turn, those subtrees contain the resources within the different departments.

The Directory tree is made up of objects. These objects represent network entities and are of two basic types: leaf objects and container objects. In the tree representation, container objects can hold leaf objects and other container objects. Container objects are Country objects, Organization objects, or Organizational Unit objects. Leaf objects usually represent a network resource such as a user or a printer and can hold no other object.

If, for example, Acme used the Country Organizational Unit to organize its tree by country, the country organizational unit would be located between the root and the Organization objects.

An object's name context is a list of these containers between the object and [Root]. This context, or name, describes its position in the NDS tree.

Self-Check

1. What does a Directory use to keep track of information?
 - a. Branches
 - b. Leafs
 - c. Trees
 - d. SQL database
2. The Directory Tree is made up of?
 - a. objects
 - b. predicates
 - c. binary digits
 - d. LDAP searches
3. What is the top of a directory tree called?
 - a. leaf
 - b. OU
 - c. ROOT
 - d. Attribute

4. What do the objects in the directory tree represent?
- a. containers
 - b. attributes
 - c. network entities
 - d. country objects
5. Container objects can hold _____.
- a. leaf objects
 - b. other container objects
 - c. country objects
 - d. organizational objects

Answers: 1) c. Trees
2) a. objects
3) c. ROOT
4) c. NetWork entities
5) a. leaf objects
b. other container objects
c. country objects
d. organizational objects
All answers are correct

LDAP

Lightweight Directory Access Protocol (LDAP) was primarily designed to provide an easy way to interact with directories. LDAP is a lightweight alternative to the X.500 Directory Access Protocol (DAP) for use on the Internet. It uses TCP/IP stack versus the overly complex OSI stack. It also has other simplifications, such as representing most attribute values and many protocol items as textual strings, that are designed to make clients easier to implement.

LDAP is an open standard and Directories need to support the LDAP the LDAP protocol in order to allow software developers to write applications to access the Directory via the LDAP protocol. Software developers can request or submit data through LDAP. For more information about LDAP, see <http://www.ldapzone.com>.

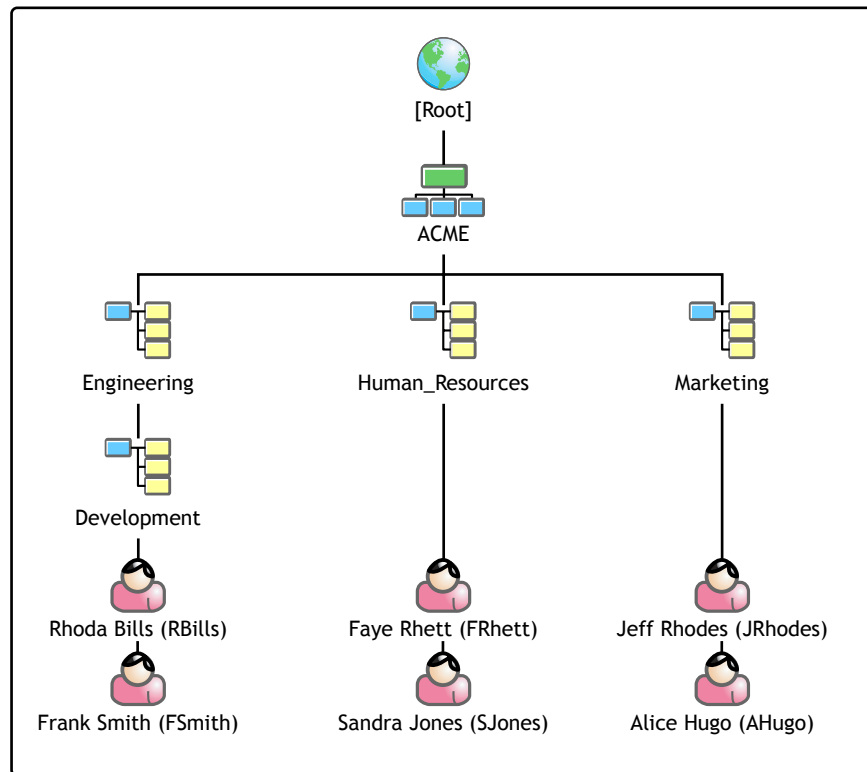


Figure 1: Example Directory Tree.

LDAP Naming Convention

The LDAP protocol defines a naming convention for objects in an LDAP Directory tree. As a software developer, you must be aware of the naming convention since you will use the LDAP naming convention in your programs.

The LDAP naming convention uses the common name of the object together with the context. The context of an object is the position of the object in the Directory tree. It allows the Directory to explicitly find the object. The common name of the object is the name of the object in the Directory. In Figure 1, user Rbills is the common name for that object. Her distinguished LDAP name that you would use if you were making an LDAP request for her object is: cn=Rbills, ou=development, ou=engineering, o=Acme. This name comes from the name attributes of her objects. The two container objects, development and engineering, are Organizational Unit objects. Acme is the Organization object.

Self-Check

1. What does LDAP stand for?
 - a. Local Distributed Access Control
 - b. Lightweight Directory Access Protocol
 - c. Lingering Distressed Applet Pattern
 - d. Lightweight Directory Applet Production
2. LDAP is an alternative to which “heavy” Directory Access Protocol?
 - a. Ethernet
 - b. Token Ring
 - c. X.500
 - d. X.255
3. Is LDAP an Open Standard?
 - a. True
 - b. False

4. Where can you learn more about LDAP?
 - a. <http://www.novell.com>
 - b. <http://www.ldapzone.com>
 - c. <http://www.amazon.com>
 - d. <http://developer.novell.com>
5. What Protocol Stack does LDAP use?
 - a. IPX
 - b. SPX
 - c. TCP/IP
 - d. Netbios

Answers:

- 1) b. Lightweight Directory Access Protocol
- 2) c. X.500
- 3) a. True
- 4) a. <http://www.novell.com>
b. <http://www.ldapzone.com>
d. <http://developer.novell.com>
- 5) c. TCP/IP

eDirectory Overview

Novell eDirectory is a distributed, hierarchical database of network information that is used to create a relationship between users and resources. It simplifies network management because network administrators can administer global networks from one location (or many) and manage all network resources as part of the eDirectory tree. User administration is simplified because the users dynamically inherit access to network resources from their placement in the eDirectory tree. For example, eDirectory enables users to dynamically inherit access to departmental resources such as applications, printers, and modems when users are placed in the department's eDirectory container. eDirectory runs on the NetWare operating system and has been ported to run on other operating systems such as Windows NT and Solaris UNIX, bringing its global administrative capabilities to NT servers. In addition, eDirectory allows you to administer intranet- and Internet-based networks.

Main Functions of eDirectory

Let's discuss some of the main functions of eDirectory.

Distributed Database

eDirectory information is typically stored not on one network server, but on several servers which are often at different locations. This allows information to be stored near users and provides efficient operation even if the users are geographically dispersed.

Hierarchical Naming Database

Names are organized in a top-down hierarchy or tree structure. This helps users find resources in a structured manner. It also enables an administrator to administer a large network by delegating portions of the tree to local administrators.

Object-Oriented Database

The entries in an eDirectory database represent network resources available on the network and are referred to as objects. An object contains information that identifies, characterizes, and locates information pertaining to the resource it represents.

Global Database

eDirectory uses a single naming system that encompasses all servers, services, and users in an internetwork. In the past, names were administered separately on each server. Now, eDirectory allows information entered once to be accessible everywhere and lets a user log in once to access diverse, geographically separated resources.

Partitioned Database

An eDirectory database can be divided into logical partitions according to business needs, network use, geographical location, access time, and other factors. These partitions can be distributed to any server represented in the directory.

Replicated Database

When an eDirectory database is distributed to multiple servers, eDirectory maintains the equality of the distributed logical partitions by distributing object information changes to the appropriate servers.

Replicated nth -level Hierarchical Database

The database provides the name space that defines the nth -level hierarchical structure.

Most importantly, eDirectory is a full service LDAP v3 compliant Directory. eDirectory manages every resource on the network. All network resources can be managed in one location with eDirectory instead of each server managing the resources connected to it.

eDirectory: A Software Developer's Perspective

The eDirectory schema is extensible, meaning that you can add new classes or attributes to the schema depending on your application needs. eDirectory provides multiple interfaces from multiple languages and platforms. LDAP is the preferred connection protocol to eDirectory so the eDirectory interfaces support LDAP. As a developer, you have access to eDirectory from VB, C, C++, Java, PHP, Perl, or any interface that has LDAP classes or support. Novell has developed LDAP libraries for C and Java. ActiveX controls have been developed for VB. LDAP classes for Java have been developed, as well as LDAP extensions for JNDI, eCommerce JavaBeans, and an LDAP JDBC driver.

As an eDirectory developer, you should be familiar with the schema of the Directory. Each API does have ways for you to programmatically see the schema of the Directory. An easier solution is to download the eDirectory Schema Reference from <http://developer.novell.com>. In this large document, you can lookup the attributes for each type of object in the Directory. This will provide a significant resource for you as you are working with the Directory schema.

Self-Check

1. Which of the following is a main function of eDirectory?
 - a. Muted Database
 - b. Cookie-Oriented Database
 - c. Elevator Naming Database
 - d. Global Database
2. The eDirectory Schema is _____?
 - a. extensible
 - b. stretchy
 - c. scaled down
 - d. enterprise Java extended
3. Which two describe what eDirectory's database is?
 - a. distributed
 - b. hierarchical
 - c. departmental
 - d. UNIX
4. What network resources can eDirectory enable users to dynamically inherit?
 - a. Printers
 - b. Applications
 - c. Modems
 - d. a, b, c and a whole lot more!

5. Names are organized in a _____ fashion in eDirectory.
- a. bottom-up
 - b. inverted trajectory
 - c. top-down
 - d. random

-
- Answers:
- 1) d. Global Database
 - 2) a. extensible
 - 3) a. distributed
b. hierarchical
 - 4) d. a, b, c, and a whole lot more!
 - 5) c. top-down

Overview of JNDI

If distributed application's components are unable to locate one another, then they can't work together. As a result, distributed applications require something to help the components to find each other. The Java Naming and Directory Interface (JNDI) provides this capability.

Naming Service

In this day and age, how many of you still get the urge to go to the library to read a book? With all the on-line items available, a lot of reading can be done from the comfort of your home via an Internet connection. If you do go to the library, finding a particular book can be easier if you use the card catalog to find out exactly where the book is. A card catalog is very similar to the idea of a naming service in computing jargon. A naming service associates names with the locations of services and with information. A naming service provides computer programs with a single location where they can find the resources they need. When a program utilizes a naming service it doesn't waste time by performing the electronic equivalent of walking up and down the aisles, and also doesn't require that the locations be hard-coded into their logic either.

Finding resources is of a particular importance in large-scale enterprise environments, where applications you build may depend on services provided by applications, written by other groups in other departments. A well thought-out naming infrastructure makes such projects possible. On the converse, the lack of one makes them impossible.

JNDI provides a common-denominator interface to many existing naming services. As such, JNDI was not designed to replace existing technology, rather, it provides a common interface to existing naming services.

Introduction to Naming Services

Figure 2, shows the organization of a generic naming service.

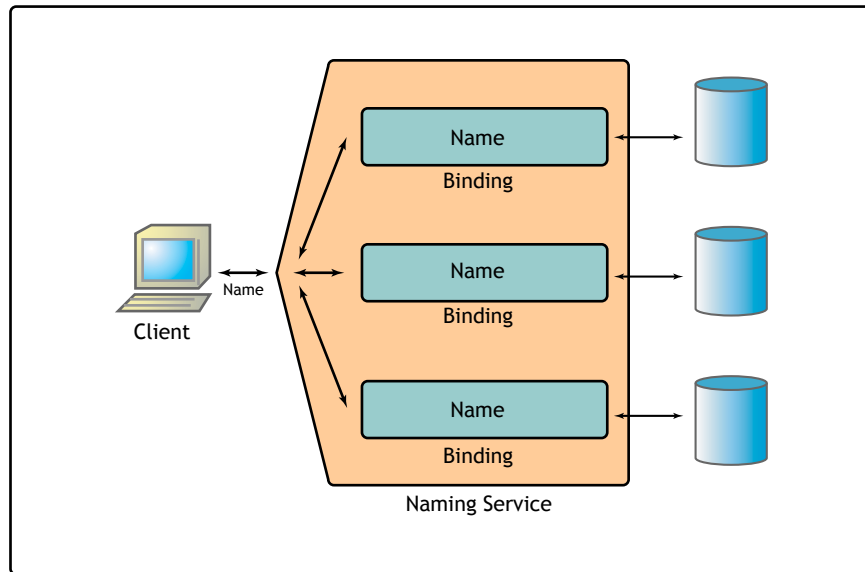


Figure 2: A Naming Service.

A naming service maintains a set of bindings. Bindings relate to the names of objects. All objects in a naming system are named in the same way, which is referred to as subscribing to the same naming convention. Clients use the naming service to locate objects by name.

There are several naming services available, a few of which will be described below. They each follow the pattern depicted in Figure 2, but differ in details.

COS (Common Object Services) Naming: The naming service for CORBA applications; allows applications to store and access references to CORBA objects.

DNS (Domain Name System): The Internet's naming services, maps people-friendly names (such as <http://www.novell.com>) into computer-friendly IP (Internet Protocol) addresses in dotted-quad notation (207.69.107.37). The DNS naming service is distributed, meaning that the service and its underlying database is spread across many hosts on the Internet.

LDAP: Developed by the University of Michigan, it is a lightweight version of the Directory Access Protocol (DAP), which in turn is a part of X.500, which is a standard for network directory services.

NIS (Network Information System) and NIS+: Network naming services developed by Sun Microsystems. Both flavors allow users to access files and applications on any host with a single ID and password.

Common Naming Features

Remember that the primary function of a naming system is to bind names to objects, or in some cases, to references to objects. In order to be a naming service, a service must at the very least provide the ability to bind names to objects and to look up objects by name.

Several naming systems do not store objects directly. Instead, they store references to objects. As an example, consider DNS. The address 207.69.107.37 is a reference to a computer's location on the Internet, not the actual computer.

JNDI provides an interface that supports all this common functionality.

Differences Among Naming Features

It is also important to understand how existing naming services differ, since JNDI must provide a workable abstraction that gets around these differences.

The biggest difference between naming services, well the most noticeable, is the different way each naming service requires names to be specified (referring to its naming convention). The following are a few examples of this concept:

1. In DNS, names are built from components that are separated by dots (“.”). They read from right to left. The name <http://www.novell.com> names a machine called “www” in the “novell.com” domain. Likewise, the name “novell.com” names the domain “novell” in the top-level domain “com.”
2. As we have discussed earlier, in LDAP, the situation is more complicated. Names are built from components that are separated by commas (“,”). Like DNS names, they read from right to left, however, components in an LDAP name must be specified as name/value pairs. The name “cn=Aaron Osmond, o=Novell, c=US” names the person “cn=Aaron Osmond” in the organization “o=Novell, c=US.” Likewise, the name “o=Novell, c=US” names the organization “o-Novell” in the country “c=US.”

As the two examples above illustrate, a naming service's naming convention alone has the potential to introduce a significant amount of the flavor of the underlying naming service into JNDI. This is not particularly a feature an implementation-independent interface should have.

JNDI solves this problem with the Name class and its subclasses and helper classes. The Name class represents a name composed of an ordered sequences of subnames, and provides methods for working with names independent of the underlying naming service.

Self-Check

1. What is a main purpose for JNDI?
 - a. help network components find each other
 - b. integrate with Beans
 - c. vacillate the equilibrium
 - d. interconnect Java leafs
2. What technology allows JNDI to be platform independent?
 - a. naming service(s)
 - b. card catalog
 - c. hard-coded for loops
 - d. interface to Beans
3. What does a naming service do?
 - a. locate objects by name
 - b. maintains a set of bindings
 - c. all objects in the naming service are named the same way
 - d. subscribes to the same naming convention
4. Which of the following are valid naming services?
 - a. COGS
 - b. COS
 - c. DNS
 - d. LDAP

5. A naming system binds names together, what else does it do?
 - a. references objects
 - b. deletes objects
 - c. renames objects
 - d. eats objects
6. Which naming system references objects?
 - a. LDAP
 - b. COGS
 - c. DNS
7. What is the biggest difference between naming services?
 - a. The different way JAVA expects the name to be spelled.
 - b. The different way each naming service requires names to be specified.
 - c. The different way that JNDI extracts attributes from an object.
 - d. The different way that Beans are imported into JNDI.
8. How are LDAP names read?
 - a. right to left
 - b. left to right
 - c. middle to right
 - d. middle to left
9. The address 207.69.107.37 is a _____ to a computer's location on the Internet, and not the actual computer.
 - a. ping
 - b. pointer
 - c. reference
 - d. dialect

10. What does the Name class provide?

- a.** methods for working with names dependent of the underlying name service.
- b.** methods for working with names independent of underlying name service.
- c.** methods that evolve dependant of the underlying name service.
- d.** classes for working with names independent of the underlying name service

-
- Answers:**
- 1) a. help network components find each other
 - 2) a. naming service
 - 3) a. ocate objects by name
 - b. maintains a set of bindings
 - c. all objects in the naming service are named the same way
 - d. subscribes to the same naming convention
 - 4) b. COS
 - c. DNS
 - d. LDAP
 - 5) a. references objects
 - 6) c. DNS
 - 7) b. The different way each naming service requires names to be specified.
 - 8) a. right to left
 - 9) c. reference
 - 10) b. methods for working with names independent of underlying name service.

JNDI Naming Explored

It is important to remember that JNDI is an interface rather than an implementation. This fact has some disadvantages – you need access to an existing naming service (such as an LDAP service) and you need to understand something about how it works in order to play with JNDI. On the other hand, it does allow JNDI to integrate seamlessly into an existing computing environment where an established naming service holds sway.

JNDI naming revolves around a small set of classes and a handful of operations. Let's take a look at them.

Context and Initial Context

The Context interface plays a central role in JNDI. A context represents a set of bindings within a naming service that all share the same naming convention. A Context object provides the methods for binding names to objects and unbinding names from objects, for renaming objects, and for listing the bindings.

Some naming services also provide subcontext functionality. Much like a directory in a file system, a subcontext is a context within a context. This hierarchical structure permits better organization of information. For naming services that support subcontexts, the Context class also provides methods for creating and destroying subcontexts.

JNDI performs all naming operations relative to a context. To assist in finding a place to start, the JNDI specification defines an Initial Context class. This class is instantiated with properties that define the type of naming service in use and, for naming services that provide security, the ID and password to use when connecting.

Let's take a look at Context's methods.

Method	Description
<code>void bind(String stringName, Object object)</code>	Binds a name to an object. The name must not be bound to another object. All intermediate contexts must already exist.
<code>Void rebind(String stringName, Object object)</code>	Binds a name to an object. All intermediate contexts must already exist.
<code>Void unbind(String stringName)</code>	Unbinds the specified object. <code>Object lookup(String stringName)</code> will return the specified object.

The Context interface also provides methods for renaming and listing bindings:.

Context Interface	Description
<code>void rename(String stringOldName, String stringNewName)</code>	Changes the name to which an object is bound.
<code>NamingEnumeration listBindings (String stringName)</code>	Returns an enumeration containing the names bound to the specified context, along with the objects and the class names of the objects bound to them.
<code>NamingEnumeration list(String stringName)</code>	Returns an enumeration containing the names bound to the specific context along with the class names of the objects bound to them.

Each of these methods has a sibling that takes a Name object instead of a String object. A Name object represents a generic name. The Name class allows a program to manipulate names without having to know as much about the specific naming service in use.

A Context Example

The example below illustrates how to connect to a naming service, list all the binding, or list a specific binding. It uses the file system service provider, which is one of the reference JNDI service-provider implementations provided by Sun. The file system service provider makes the file system look like a naming service (which it is, in many ways – filenames like `/foo/bar/baz` are names and are bound to objects like files and directories). The filesystem example was selected for this example, since not everyone has access to an LDAP server. However, further examples in the course will require a directory namespace such as LDAP.

```
import javax.naming.Context;  
import javax.naming.InitialContext;  
import javax.naming.Binding;  
import javax.naming.NamingEnumeration;
```

```

import javax.naming.NamingException;

import java.util.Hashtable;

public class Main
{
    public static void main(String [] rgstring)
    {
        try
        {
            // Create the initial context. The environment
            // information specifies the JNDI provider to use
            // and the initial URL to use (in our case, a
            // directory in URL form - file:///...).
            Hashtable hashtableEnvironment = new Hashtable();
            hashtableEnvironment.put(Context.INITIAL_FACTORY,
                "com.sun.jndi.fscontext.RefFSCContextFactory" );
            hashtableEnvironment.put(Context.PROVIDER_URL,
                rgstring[0]);

            Context context = new InitialContext(hashtableEnvironment);
            // If you provide no other command line arguments,
            // list all of the names in the specified context and
            // the objects they are bound to.

            if(rgstring.length == 1)
            {
                NamingEnumeration namingenumeration =
                    context.listBindings("");
                while(namingenumeration.hasMore())
                {
                    Binding binding =
                        (Binding)namingenumeration.next();
                    System.out.println(binding.getName() +
                        " " + binding.getObject());
                }
            }
            // Otherwise, list the names and binding for the
            // specified arguments.
            else
            {
                for(int i = 1; i < rgstring.length; i++)
                {
                    Object object = context.lookup(rgstring[i]);
                    System.out.println(rgstring[i] + " " + object);
                }
            }
            context.close();
        }
        catch(NamingException namingexception)
        {
            namingexception.printStackTrace();
        }
    }
}

```

The program in the listing above first creates an initial context from the specified JNDI provider (in this case, Sun's filesystem provider) and a URL specifying a local directory. If no additional command-line arguments are specified, the program lists the objects and names of every entity in the specified directory. Otherwise, it lists the objects and names of only those items specified on the command line.

Returning to Directory Services

A directory service provides a way to manage the storage and distribution of shared information. Such information can range from the e-mail address and phone numbers of a company's employees, to the IP addresses and print capabilities of a department's printers, to the configuration information for a suite of application servers. Figure 3 illustrates a generic directory service.

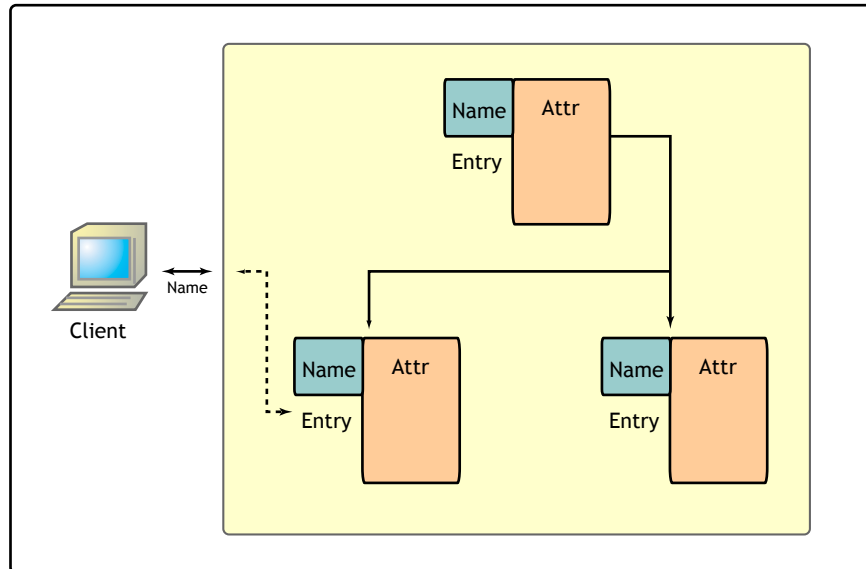


Figure 3: A Generic Directory Service.

A directory service manages a directory of entries. A directory entry can refer to a person, place, service, or almost any other concrete or abstract concept. An entry also has attributes associated with it; an attribute consists of a name or identifier and one or more values. These attributes describe the entry, and the exact set of attributes depends on the type of the entry. For example, the entry for an individual might have the following attributes:

Name: David Coperfield
Address: 123 Somewhere in Vegas
Email: David@magicismylife.com
Email: DCopperfield@figureitout.com

Directory services are simple databases. Like their relational cousins, many common directory services provide search and filter functionality. Instead of locating an entry point only by name, these directory services allow you to locate entries based on a set of search criteria.

Naming services and directory services are logical partners. In fact, most existing products provide both sets of functionality. Naming services provide name-to-object mapping, and directory services provide information about the objects and tools for searching for them.

There are a number of existing directory service products; LDAP is the most common. LDAP provides both naming and directory functionality. Numerous commercial implementations exist, including eDirectory. Some are even free such as OpenLDAP (<http://openldap.org/>).

Bringing JNDI Into the Directory Picture

JNDI directory-service support is both comprehensive and powerful. It adds advanced functions, like storing and retrieving serialized class instances, to searching and other components of the basic suite of direct functions. With this in mind, let's examine the `DirContext` class, which is the heart of JNDI directory services.

Working With Attributes

The `DirContext` class is a subclass of the `Context` class described last month. It provides all of the standard naming service functionality, and can also work with attributes and search for directory entries.

Let's take a look at the methods of `DirContext` that extend those methods provided by the `Context` class.

The `bind()` method

```
void bind (  
    String stringName,  
    Object object,  
    Attributes attributes)
```

The `bind()` method binds a name to an object and stores the specified attributes with that entry. This operation generally tries to preserve existing attributes in cases in which that makes sense. Specifically, if attribute is null and object is an instance of the `DirContext` class, the resulting binding will retain the attributes originally associated with object.

The `rebind()` method

```
void rebind(  
    String stringName,  
    Object object,  
    Attributes attributes)
```

The `rebind()` method binds a name to an object and stores the specified attributes with that entry. The previous binding is replaced. As is the case with `bind()`, this operation tries to preserve existing attributes in cases in which that would make sense.

The `createSubcontext()` method

```
DirContext createSubcontext(  
    String stringName,  
    Attributes attributes)
```

The `createSubcontext()` method creates a new subcontext and binds a name to it, and then stores the specified attributes with that entry. If attributes is null, this method works in exactly the same fashion as the like-named method of the `Context` class.

Let's next consider those of `DirContext`'s methods that do not extend methods provided by the `Context` class, providing the tools for working with attributes instead.

The `getAttributes()` methods

The class provides two flavors of this method:

```
Attributes getAttributes(  
    String stringName)
```

And:

```
Attributes getAttributes(  
    String stringName,  
    String[] rgstringAttributeNames)
```

The `getAttributes` methods return the attributes associated with the specified entry. The `Attributes` class represents a collection of attributes; it contains instances of the `Attribute` class, which by itself represents a single attribute. The first method returns all attributes, and the second returns the attributes names in the supplied array of attribute names.

The `modifyAttributes` methods
`modifyAttributes` comes in two flavors as well:

```
void modifyAttributes(  
    String stringName,  
    int nOperation,  
    Attributes attributes)
```

And:

```
void modifyAttributes(  
    String stringName,  
    ModificationItem[] rgmodificationitem)
```

The `modifyAttributes` methods modify the attributes associated with the specified entry. The permitted operations are `ADD_ATTRIBUTE`, `REPLACE_ATTRIBUTE`, and `REMOVE_ATTRIBUTE`. The method modifies several attributes in the same way, while the second performs a series of modifications on one or more attributes.

As with the `Context` class, each of the methods above has a variant that takes a `Name` object rather than a `String` object.

Searching

In order to make use of directory service functionality, it is necessary to have some way to search the contents of a directory service. The `DirContext` class provides two general models by which searches may be conducted.

Searching by attribute name

There are two ways to conduct a search following this model:

```
NamingEnumeration search(  
    String stringName,  
    Attributes attributesToMatch)
```

And:

```
NamingEnumeration search(  
    String      stringName,  
    Attributes  attributesToMatch,  
    String[]    rgstringAttributesToReturn)
```

In this first model, the search occurs within a single context for entries that contain a specific set of attributes. For the entities that match, the search retrieves either the entire set of attributes (if the search is implemented using the first block of code above) or a select set of attributes (if the search is implemented using the second).

Searching by RFC 2254 filter

There are two ways to perform this search:

```
NamingEnumeration search(  
    Name          stringName,  
    String        stringRFC2254Filter,  
    SearchControls searchcontrols)
```

And:

```
NamingEnumeration search(  
    Name          stringName,  
    String        stringRFC2254Filter,  
    Object[]      stringRFC2254FilterArgs,  
    SearchControls searchcontrols)
```

In the second model, the search occurs within a context for entries that satisfy a search filter. RFC2254 (which describes a string representation for LDAP search filters, defines the format of the filter.

An instance of the SearchControls class controls key aspects of the search:

```
SearchControls(  
    int      nSearchScope,  
    long     nEntryLimit,  
    int      nTimeLimit,  
    String[] rgstringAttributesToReturn,  
    boolean  boolReturnObject,  
    boolean  bool DereferenceLinks)
```

The constructor above lists all of the aspects of a search that a SearchControls instance affects. Corresponding accessors (get and set methods) also exist. The following table provides a short description of each.

Method	Description
nSearchScope	Sets the scope of the search of either the object (OBJECT_SCOPE), the object and the level immediately below it (ONELEVEL_SCOPE), or the object and its entire subtree (SUBTREE_SCOPE).
nEntryLimit	Sets the maximum number of entries that the search will return.
nTimeLimit	Sets the maximum number of milliseconds that the search will run.
rgstringAttributesToReturn	Determines which attributes should be returned along with the entries returned by the search.
boolReturnObject	Determines whether or not the objects bound to selected entries should be returned along with the entries returned by the search.
boolDereferenceLinks	Determines whether or not links should be dereferenced links (or followed to their ultimate destination) during the search. A link references another directory entry and can span multiple naming systems. The underlying JNDI service provider may or may not provide support for links.

With these methods in the table, they also have a variant that takes a Name object rather than a String.

Hopefully you have a solid understanding of both naming and directory services, and a general understanding of JNDI. If not, please review the Directory Concepts, LDAP, eDirectory Overview, and Overview of JNDI sections.

Self-Check

1. JNDI is a Java implementation.
 - a. True
 - b. False
2. JNDI is a Java interface.
 - a. True
 - b. False
3. JNDI naming revolves around a small set of classes and a handful of operations.
 - a. True
 - b. False

4. What does a context represent?
 - a. a set of interfaces that emit light
 - b. a servlet that enables a leaf tree
 - c. a set of bindings within a naming service that share the same naming convention
 - d. a set of bindings within a naming service that do not share the same naming convention
5. Which are methods of the Context Class?
 - a. bind
 - b. rebind
 - c. unbind
 - d. all the above
6. What are the two general models for searching provided by the DirContext class?
 - a. Searching by attribute name
 - b. Searching by class name
 - c. Searching by RFC 2254 filter
 - d. a and c

-
- Answers:
- 1) b. False
 - 2) a. True
 - 3) a. True
 - 4) c. a set of bindings within a naming service that share the same naming convention
 - 5) d. all the above
 - 6) d. a and c

JNDI a History

JNDI plays a role in a number of Java technologies. Three of them are as follows:

1. JDBC (the Java Database Connectivity package)
2. JMS (the Java Messaging Service)
3. EJB (the Enterprise JavaBeans)

JDBC— The Java technology for relational databases. JNDI first appeared in the JDBC 2.0 Optional Package in conjunction with the DataSource interface. A DataSource instance, as its name implies, represents a source of data, often from a database, but not always. A DataSource instance stores information about a data source, such as its name, the driver to load and use, and its location, and allows an application to obtain a connection to the data source without regard to the underlying details. The JDBC specification recommends using JNDI to store DataSource objects.

JMS— The Java technology for messaging. The JMS specification describes administered objects – objects that contain JMS configuration information and are used by JM to locate specific message queues and topics. As is the case with JDBC, the specification recommends locating JMS administered objects via JNDI.

EJB— All enterprise beans publish a home interface via JNDI. This is the single location though which clients locate a specific enterprise bean.

So, if this is the case, why is JNDI so highly regarded? For one reason, JNDI promotes the notion of a centrally managed information source, which is a key requirement for enterprise applications. A centrally managed information source is easier to administer than a distributed collection of information sources. It's also simpler to clients to locate needed information if they only have to look in one place.

Also, JNDI's ability to directly store Java objects allows it to integrate almost transparently into Java applications.

JNDI Provider

To use JNDI, you need a naming and directory service and JNDI service provider. Sun supplies several providers of common naming and directory services such as COSnaming, NIS, RMI, LDAP and more. For the sake of this course and since eDirectory supports it, I have chosen LDAP.

LDAP has the advantage of being widely implemented (eDirectory) in commercial and open-source forms and also being quite easy to use. Its features are also well supported by Sun's LDAP service provider and JNDI.

In addition to eDirectory, there are other options for LDAP providers. Numerous LDAP implementations are available. Many are commercial products such as eDirectory, Netscape Directory Server and IBM's Secure Way Directory. Some are packaged as a part of a larger offering (eDirectory is part of the NetWare server). If you have access to such an implementation of eDirectory or another commercial LDAP provider, then you can skip the next section on OpenLDAP, if not, then OpenLDAP may be a good solution for you.

OpenLDAP

OpenLDAP is a freely available implementation of LDAP that is based on University of Michigan's LDAP reference implementation. OpenLDAP is available from the OpenLDAP Foundation (<http://www.openldap.org>). OpenLDAP's license is referred to as an "artistic license," which means that open LDAP is free (or open source) software. Prepackaged files are available for various flavors of Linux as well as BSD Unix. Work is currently underway on a port for Windows NT.

Connecting to a JNDI Context

Before you can do anything with JNDI, you need an initial context. All operations are performed relative to the context or one of its subcontexts. Follow these steps to setup a context:

1. Select a LDAP service provider. If you choose to use the OpenLDAP or some other LDAP implementation, Sun supplies a reference LDAP service provider (<http://java.sun.com/products/jndi/>). Add the name of the service provider to the set of environment properties, which is stored in a hashtable instance.

```
Hashtable hashtableEnvironment = new Hashtable();
hashtableEnvironment.put (
    Context.INITIAL_CONTEXT_FACTORY,
    "com.sun.jndi.ldap.LdapCtxFactory");
```

2. Add any extra information the service provider requires. For LDAP, that includes the URL that identifies the service, the root context, and the name and password to connect with:

```
// the service: ldap://localhost:389/
// the root context: dc=etcee,dc=com
hashtableEnvironment.put(
    Context.PROVIDER_URL,
    "ldap://localhost:389/dc=etcee,dc=cpm");
hashtableEnvironment.put(
    Context.SECURITY_PRINCIPAL,
    "name");
hashtableEnvironment.put(
    Context.SECURITY_CREDENTIALS,
    "password");
```

3. Finally get the initial context. If you just intend to perform naming operations, you will only need a Context instance. If you intend to perform a directory operation as well, you will need a DirContext instance instead. Not all providers supply both:

```
Context context = new
InitialContext(hashtableEnvironment);
```

Or for a directory:

```
DirContext dircontext = new
InitialDirContext(hashtableEnvironment);
```

Now let's look at how applications store objects to and retrieve objects from JNDI.

Dealing With Objects

The ability to store Java objects is useful: object storage provides persistence and allows objects to be shared between applications or between different executions of the same application. From the standpoint of the code involved, object storage is quite easy:

```
context.bind("name", object);
```

The bind() operation binds a name to a Java object. It is permissible for the bind() operation to store either a snapshot of the object or a reference to a “live” object, for example.

Be aware that the bind() operation throws a NamingException if an exception occurs during the execution of the operation.

Now let’s take a look at the bind() operation’s complement – lookup():

```
Object object = context.lookup("name");
```

The lookup() operation retrieves the object bound to the specified name. Just as with bind(), the lookup() operation throws a NamingException if an exception occurs during the execution of the operation.

Storing Objects

What does it mean to store an object in a JNDI naming and directory service? We have already mentioned that the exact semantics of the bind() and lookup() operations are not tightly defined; it is up to the JNDI service provider to define their semantics.

According to the JNDI specification, service providers are encouraged (but not required) to support object storage in one of the following formats:

- Serialized data
- Reference
- Attributes in a directory context

If all JNDI service providers support these standard mechanisms, Java programmers are free to develop generic solutions that work even when the underlying service provider layer changes.

Each of the methods above has advantages and disadvantages. The best method will depend on the requirements of the application under development.

Let’s look at an example of each:

Serialized Data

The most obvious approach to storing an object in a directory is to store the serialized representation of an object. The only requirement is that the object’s class implement the Serializable interface.

When an object is serialized, its state becomes transformed into a stream of bytes. The service provider takes the stream of bytes and stores it in the directory. When a client looks up the object, the service provider reconstructs it from the stored data.

The following code shows how to bind a LinkedList to an entry in a JNDI service:

```
// create a linked list
LinkedList linkedlist = new LinkedList();
.
.
.
// bind
context.bind("cn=novell", linkedlist);
.
.
.
// lookup
linkedlist = (LinkedList)context.lookup("cn=novell");
```

Reference

Sometimes it is not appropriate or not possible to serialize an object. If the object provides a service on a network, for example, it doesn't make sense to store the state of the object itself. We are interested in the information necessary to find and communicate with the object.

An example is a connection to an external resource (one outside the scope of the Java Virtual Machine) such as a database or file. It clearly doesn't make sense to try to store the database or the file itself in the JNDI service. Instead, we want to store the information necessary to reconstruct the connection.

In this case the programmer should either bind a Reference instance that corresponds to the object or have the object's class implement the Referenceable interface (in which the object generates and provides a Reference instance when requested by the service provider).

The Reference instance contains enough information to recreate the reference. If a reference to a file was stored, the reference contains enough information to create a File object that points to the correct file.

Attributes in a Directory Context

If you are using a service provider that provides directory functionality instead of only naming functionality, you can also store the object as a collection of attributes on a DirContext object (a DirContext instance differs from a Context instance in that it may have attributes).

To use this method, you must create objects that implement the DirContext interface and contain the code necessary to write their internal state as an Attributes object. You must also create an object factory to reconstruct the object. This approach is useful when the object must be accessible by nonJava applications.

Up next, a sample application.

Note: I am borrowing this application framework from Todd Sundsted's article series, JNDI Overview Parts 1, 2, and 3 available at <http://java.sun.com>.

Self-Check

1. What three roles does JNDI play in Java Technologies?
 - a. COGS, LDAP, FIGS
 - b. JDBC, JMS, EJB
 - c. JDBC, LDAP, Beans
 - d. FIGS, JMS, EJB
2. What technology is JDBC?
 - a. Java connector technology
 - b. Beans technology
 - c. JMS Exploited Beans technology
 - d. relational database technology
3. What technology is JMS?
 - a. Java connector technology
 - b. relational database technology
 - c. messaging technology
 - d. RFC 2254 Filter technology

4. What is EJB?
 - a. Enterprise Java Beans
 - b. Enterprise on Star Trek
 - c. Enterprise Jakarta Beans
 - d. Enterprise JNDI Beans
5. JNDI's ability to directly store Java objects allows it to integrate almost transparently into Java applications.
 - a. True
 - b. False

Answers: 1) b. JDBC, JMS, EJB
2) d. relational database technology
3) c. messaging technology
4) a. Enterprise Java Beans
5) a. True

The Doc-u-matic, a JNDI Application

At this point in the course, you should feel comfortable with naming and directory services and the operations they support. We will be working with a document publication and distribution application called Doc-u-Matic that illustrates JNDI's support for the following:

- Centralized information administration
- Network-wide information distribution
- Object persistence

Doc-u-Matic is first-and-foremost a demonstration of JNDI-supported object persistence. Let's first start with a review of JNDI's support for stored objects.

JNDI's Support for Stored Objects

You may recall that there are three techniques for storing Java objects in a JNDI service: as serialized data, as a reference to an object, and as the attributes on a directory context. Storing an object as serialized data is the simplest of the three techniques. Storing an object as a reference is useful in situations in which it doesn't make sense (or isn't possible) to store actual objects. Finally, storing objects as attributes on a directory context is useful when other, non-Java applications need access to the object's information. The application we are going to look at uses two methods of object storage.

Building Blocks

Figure 4, below illustrates the Doc-u-Matic utility from a functional perspective. Doc-u-Matic consists of three major functional units: the JNDI service, a library, and one or more clients.

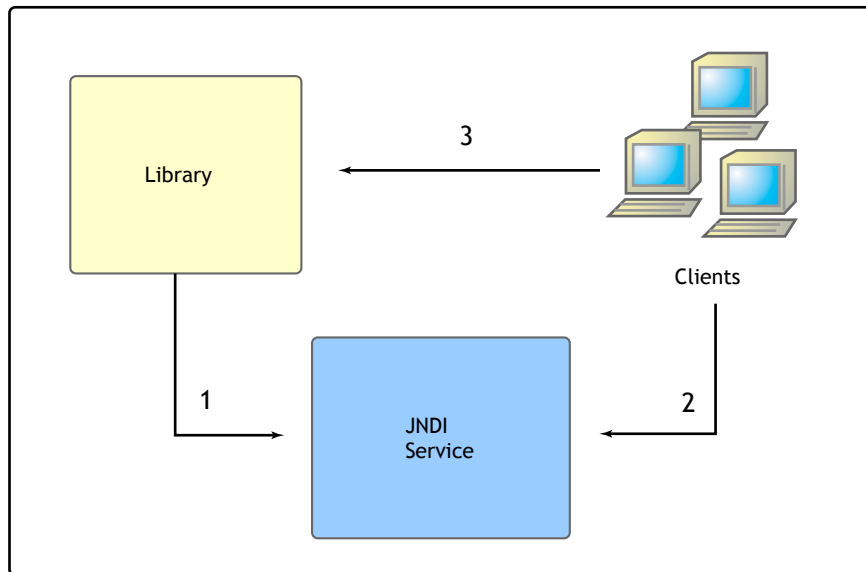


Figure 4: *Functional View of Doc-u-Matic.*

A library is a place to which you publish objects and from which you retrieve them. An object can be any Java instance, as long as it supports one of the storage methods mentioned above. It can be a String, a JavaBean, and so on.

JNDI plays two roles in Doc-u-Matic. It provides a single, well-known location where clients can locate the library service (standard address-book functionality), and it supports the implementation of the library service itself. On the latter point, it is worth noting that nothing in the design of a library implies that it has to be implemented on top of JNDI. You could implement a library on top of any technology that provides support for publishing and retrieving information, including HTTP, JDBC, NNTP, or IMAP. Best of all, the clients would never know the difference.

Preparation

Doc-u-Matic assumes you have obtained, or have access to, and have configured an LDAP service (eDirectory for example). The application should support any naming and directory service that provides a JNDI service provider. However, I will assume that you are using LDAP.

Before you proceed, make sure you have obtained, installed, and configured the following:

- An LDAP implementation
- Sun's JNDI reference implementation and the LDAP service provider.
- Create the initial context that the application will connect to. The following is assumed:

Ou=HowTo,o=JavaWorld

Proper Usage

Before we jump into the code, let's stop for a moment and look at how to use Doc-u-Matic. There are two usage roles: the administrator and the user. The administrator creates or deploys the library and publishes objects of various types in the library. The administrator also creates and distributes a properties file that contains all of the information necessary to connect to and use the library. Users, on the other hand, retrieve objects from the library.

Administrator's Role

Let's begin by assuming the role of the administrator. The deployment tool is named JNDIDeploy. Before you deploy a library, you must create a properties file that contains the information necessary to connect to an initial context. The properties file must also contain the name that the JNDILibrary object will be bound to. Take a look at the following properties file:

```
# DEPLOYMENT PROPERTIES
# This properties file contains all of the information necessary to
# find and connect to the JNDI service that holds the library and all
# published objects.
java.naming.factory.initial = com.sun.jndi.ldap.LdapCtxFactory
java.naming.provider.url = ldap://localhost:389/ou=HowTo,o=JavaWorld
library.name = cn=library
```

The library deploys as follows (I will assume that you have already setup the classpath to point to the jndi.jar and ldap.jar JAR files):

Java JNDIDeploy <properties file>

Once you, as administrator, have deployed the library, you must distribute a properties file to all users. The client applications all read a properties file pointed to by a URL. As a result, the easiest way to distribute the file is to put it on a Web server and distribute the URL of the properties file.

Once you have deployed a library, you can publish objects to the library. By objects, I mean instances of Java classes. Objects play the role of abstract containers of information. As such, the simplest object is probably an instance of the String class. There are a few stipulations on publishable objects; they must be instantiable via the Beans.instantiate() method, and they must be storable via JNDI. Objects are published as follows:

```
java Publish <properties URL> [name class]...
```

The Publish command requires the URL of the properties file mentioned above. It also accepts any number of additional pairs of arguments. The first value in each pair is the name of the object. The second is the name of the class (including package information) that will be instantiated and stored. The name must conform to whatever naming policy the JNDI service provider requires. In the case of LDAP, names will be of the form:

```
<key>=<value>
```

User's Role

With the user's role it is time to retrieve some of the objects that have been published via the administrator. As a user, the only facts we have to know to retrieve an object are its name and the URL that describes the location of the library. We don't have to know how the library is implemented. This is especially useful if we, as users, must write programs to use the library. As you will see a little bit later, the code required to use a library is very small indeed.

There is a small client program that is designed to retrieve objects from the library. Objects are retrieved as follows:

```
java Client <properties URL> [name]...
```

The Client command requires the URL of the properties file. It also accepts any number of additional arguments specifying the names of objects to retrieve from the library.

Code

Figure 5, below, shows the six classes that form the core of Doc-u-matic. We will discuss each briefly and then present important parts of the code.

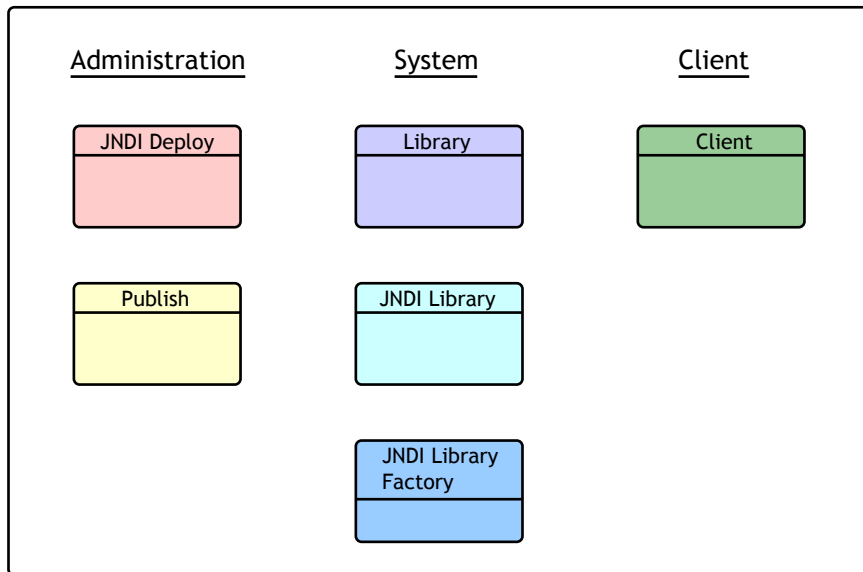


Figure 5: *Doc-u-Matic's Six Core Classes.*

The Library interface defines the methods that all libraries must provide. For the sake of simplicity, it requires only two methods: one to publish objects and one to retrieve objects. Full-featured libraries would provide search functionality as well.

```
// Publishes an object in the library
//
// @param object the object
// @param stringName the name of the object
// @param map a map containing the object's attributes
//
public void publish(
    Object    object,
    String    stringName,
    Map      map);

// Retrieves a copy of a published object from the library
// and restores it if necessary.
//
// @param stringName the name of the object
// @returns the object, or null
//
public Object retrieve(String stringName);
```

The JNDILibrary class provides a JNDI-based implementation of the Library interface. The publish() and retrieve() methods demonstrate how to use JNDI to bind objects to look up objects from a directory service.

```

// Publishes an object in the Library
//
// @param object the object
// @param stringName the name of the object
// @param map a map containing the object's attributes
//

public void publish(
    Object object,
    String stringName,
    Map map);
{
    try
    {
        // Use the properties to establish an initial directory context.
        DirContext dircontext = new InitialDirContext(m_properties);

        // Create an iterator that contains all the entries in the map.
        Iterator iterator = map.entrySet().iterator();

        // For each entry, create an attribute.
        BasicAttributes basicattributes = new BasicAttributes();
        while(iterator.hasNext())
        {
            Map.Entry entry = (Map.Entry) iterator.next();
            Basicattributes.put(entry.getKey().toString(),
entry.getValue().toString());
        }

        // Bind the object to the specified name.
        dircontext.rebind(stringName, object, basicattributes);
        // Close the context.
        Dircontext.close();
    }
    catch (Exception exception)
    {
        exception.printStackTrace();
    }
}

// Retrieves a copy of a published object from the library
// and restores it if necessary.
//
// @param stringName the name of the object
// @returns the object, or null
//
public Object retrieve(String stringName)
{
    Object object = null;
    Try
    {
        // Use the properties to establish an initial directory context.
        DirContext dircontext = new InitialDirContext(m_properties);
        // Look up the object using the specified name.
        object = dircontext.lookup(stringName);
        // Close the context.
        Dircontext.close();
    }
    catch(Exception exception)

```

```

    {
        exception.printStackTrace();
    }
    return object;
}

```

The `getReference()` method demonstrates how to create a `Reference` instance that represents the current instance of a class. Instead of being stored as serialized data, classes that implement the `Referenceable` interface are stored indirectly, via `Reference` instances. Notice how, in the example below, the properties are transformed and stored as a series of bytes. Typically, enough of an object's state must be stored to create a working copy of the object when the object is looked up in a directory service.

```

// Gets the reference for the library
//
// @returns the reference
//
public Reference getReference()
{
    Reference reference = null;
    Try
    {
        // Store the properties as an array of bytes.
        ByteArrayOutputStream byteArrayOutputStream = new
            ByteArrayOutputStream();
        M_properties.store(byteArrayOutputStream, null);
        // Create a reference to the library.
        reference = new Reference(
            JNDILibrary.class.getName(),
            New BinaryRefAddr("properties",
            byteArrayOutputStream.toByteArray()),
            JNDILibraryFactory.class.getName(),
            null);
    }
    catch(Exception exception)
    {
        exception.printStackTrace();
    }
    return reference;
}

```

JNDILibraryFactory.java

The `JNDILibraryFactory` class is used on the client-side to create an instance of the `JNDILibrary` class when a JNDI library is looked up in a directory service. `JNDILibraryFactory`'s primary operation: transform the stored binary property-file information into live property information. This transformation allows the new instance to function identically to the version that was stored:

```

// Creates an object instance given a reference.
//
// @param object a reference
// @param name a name
// @param context the context
// @param hashtable the environment
//
// @returns an object, or null
//
public Object getObjectInstance(
        Object    object,
        Name      name,
        Context   context,
        Hashtable hashtable)
throws Exception
{
    // Checks whether or not the object is an instance of a reference.
    if (object instanceof Reference)
    {
        Reference reference = (Reference)object;
        // Checks whether or not it is a valid reference for this
        // factory.
        if (reference.getClassName().equals(JNDILibrary.class.getName()))
        {
            // Gets the stored payload.
            RefAddr refaddr = reference.get("properties");
            if (refaddr != null)
            {
                if (refaddr instanceof BinaryRefAddr)
                {
                    BinaryRefAddr binaryrefaddr =
                        (BinaryRefAddr)refaddr;
                    // Recreates the stored properties.
                    Byte[] rgb = (byte []) binaryrefaddr.getConteibt();
                    ByteArrayInputStream bytearrayinputstream =
                        New ByteArrayInputStream(rgb);
                    Properties properties = new Properties();
                    properties.load(bytearrayinputstream);
                    // Creates a new JNDI library with the
                    // stored properties.
                }
            }
        }
    }
    return null;
}

```

The getObjectInstance() and the getReference() methods, provided by the JNDILibrary class, work as a pair.

JNDIDeploy.java

JNDIDeploy class deploys or creates a new library. The body implements as close to a textbook example of binding via JNDI as you are likely to find:

```

// Deploys a JNDILibrary instance.
//
// This class should be run as an application. It deploys a
// JNDILibrary instance. It assumes the existence of a
// properly configured JNDI service (typically, but not necessarily,
// running LDAP).
//
// The application requires a single command-line argument:
// the name of a file containing the deployment properties. The
// properties file may contain any valid JNDI property, as well as the
// property "library.name", which must contain the name of the
// library.
//
public static void main (String[] rgstring)
{
    if (rgstring.length != 1)
    {
        System.out.println("Usage: java JNDIDeploy ");
        System.exit(-1);
    }
    try
    {
        Properties properties = new Properties();
        // Load the properties from the specified file.
        properties.load(new FileInputStream(rgstring[0]));
        // Use the properties to establish an initial directory context.
        DirContext dircontext = new InitialDirContext(properties);
        // Create the JNDI library instance and initialize it with the
        // same set of properties (it will publish objects in the same
        // context as itself).
        JNDILibrary jndilibrary = new JNDILibrary(properties);
        // Get the name of the library.
        String stringName = properties.getProperty("library.name");
        // Bind the library to the specified name.
        dircontext.rebind(stringName, jndilibrary, null);
        // Close the context.
        dircontext.close();
    }
    catch (Exception exception)
    {
        exception.printStackTrace();
        System.exit(-1);
    }
}

```

Publish.java

The Publish class publishes one or more objects in a library. Once again, you are not likely to find a more textbook example of how to look up an object via JNDI – in this case the library. Once you have retrieved the library, it can be used to push objects. Notice how the code doesn't have to know anything about how the library is implemented. You'll also notice that the Beans.instantiate() method instantiates the objects, which facilitates the storage of objects that have been put together in the JavaBeans development tool and serialized to disk:

```

// Publishes an object.
//
public static void main(String[] rgstring)
{
    if(rgstring.length < 1)
    {
        System.out.println("Usage: java Publish *lt:URL> [name class]...");
        System.exit(-1);
    }
    try
    {
        Properties properties = new Properties();

        // Load the properties from the specified URL.
        Properties.load(new URL(rgstring[0].openStream()));

        // Use the properties to establish an initial directory context.
        DirContext dircontext = new InitialDirContext(properties);

        // Get the name of the library.
        String stringName = properties.getProperty("library.name");

        // Look up a library using the specified name.
        Library library = (Library) dircontext.lookup(stringName);

        // Close the context.
        Dircontext.close();

        // For each command-line argument, instantiate the specified
        // object, and publish it to the library.
        for (int i = 1; i < rgstring.length; i++)
        {
            Object object = Beans.instantiate(null, rgstring[i + 1]);
            Library.publish(object, rgstring[i], new HashMap());
            i++;
        }
    }
    catch (Exception exception)
    {
        exception.printStackTrace();
        System.exit(-1);
    }
}

```

Client.java

The Client class is almost identical to the Publish class, except that it retrieves rather than publishes them.

```

// A Simple Client
//
// Demonstrates how to connect to and retrieve from a Library instance via JNDI.
//
public static void main(String[], rgstring)
{
    if(rgstring.length < 1)
    {

```

```

        System.out.println("Usage: java Client <URL> [name] ...");
        System.exit(-1);
    }
    try
    {
        Properties properties = new Properties();

        // Load the properties from the specified URL.
        properties.load(new URL(rgstring[0].openStream()));

        // Use the properties to establish an initial directory context.
        DirContext dircontext = new InitialDirContext(properties);

        // Get the name of the library.
        String stringName = properties.getProperty("library.name");

        // Look up a library using the specified name.
        Library library = (Library)dircontext.lookup(stringName);

        // Close the context.
        Dircontext.close();

        // For each command-line argument, retrieve the specified object
        // from the library.
        for (int i = 1; i < rgstring.length; i++)
        {
            Object object = library.retrieve(rgstring[i]);
            // If the object is restorable, restore it.
            if(object instanceof Restorable)
            {
                Restorable restorable = (Restorable)object;
                Restorable.restore();
            }
        }
    }
    catch (Exception exception)
    {
        exception.printStackTrace();
        System.exit(-1);
    }
}

```

Doc-u-Matic Tutorial

After going through this short tutorial and sample JNDI application, you should have a much clearer understanding of naming and directory services, JNDI, its capabilities, and the flexibility it brings to JNDI-enabled enterprise applications. JNDI already plays an important role in several key Java APIs, but this is beyond the scope of this course. You can bet one thing though – you can expect JNDI’s role to continually expand.

Self-Check

1. 1. The Doc-u-matic application illustrates JNDI's support for which of the following?
 - a. Centralized information administration
 - b. Network-wide information distribution
 - c. Object persistence
 - d. LDAP information exchange
2. Which is a technique for storing Java objects?
 - a. serialized data
 - b. stereo data
 - c. binary data
 - d. encrypted data
3. What is another technique(s) for storing Java objects?
 - a. reference to a class
 - b. reference to an object
 - c. attributes on a directory leaf
 - d. attributes on a directory context
4. What are the two usage roles for Doc-u-matic?
 - a. User and Supervisor
 - b. Administrator and User
 - c. Player and User
 - d. Player and Administrator
5. What is the Doc-u-matic's user role?
 - a. retrieve some of the objects that have been published via the administrator
 - b. retrieve some of the objects that have been published via the client
 - c. store some of the objects that have been published via the administrator
 - d. store some of the objects that have been published via the client

-
- Answers:**
- 1)
 - a. Centralized information administration
 - b. Network-wide information distribution
 - c. Object persistence
 - 2)
 - a. serialized data
 - 3)
 - b. reference to an object
 - d. attributes on a directory context
 - 4)
 - b. Administrator and User
 - 5)
 - a. retrieve some of the objects that have been published via the administrator

Resources

- To download the complete source for Doc-u-Matic, see:
<http://www.javaworld.com/javaworld/jw-03-2000/howto/jw-0331-howto.zip>
- To download the LDIF file, see:
<http://www.javaworld.com/javaworld/jw-03-2000/howto.ldif>
- For JNDI information, service providers, and so on, go to:
<http://java.sun.com/products/jndi>
- An alternate, and very good JNDI Tutorial can be found at:
<http://java.sun.com/products/jndi/tutorial/index.html>
- For the schema used to represent Java objects in an LDAP directory, see: <http://www.ietf.org/rfc/rfc2713.txt>.

Using Novell's LDAP Extensions and Controls for JNDI

The LDAP Extensions and Controls for JNDI are LDAP v3-compliant and include support for the virtual list views and server-side sorting controls available on eDirectory. Because it uses LDAP, it has no dependencies on the Novell Client software.

For information on using the controls and extensions with Novell eDirectory, see LDAP Control Support and LDAP Extension Support in the NDS and LDAP Integration Guide at:

<http://developer.novell.com/ndk/doc/extjndi/index.html?page=/ndk/doc/extjndi/ejndienu/data/a9imlo4.html>.

```
// Sample code file: CreateNamingContext.javaFirst of all, let's look at some
// sample code for creating a context:
// Warning: This code has been marked up for HTML
/*****
* Copyright (c) 2002 Novell, Inc. All Rights Reserved.
* THIS WORK IS SUBJECT TO U.S. AND INTERNATIONAL COPYRIGHT LAWS
* AND TREATIES. USE AND REDISTRIBUTION OF THIS WORK IS SUBJECT TO
* THE LICENSE AGREEMENT ACCOMPANYING THE SOFTWARE
* DEVELOPMENT KIT (SDK) THAT CONTAINS THIS WORK. PURSUANT TO
* THE SDK LICENSE AGREEMENT, NOVELL HEREBY GRANTS TO
* DEVELOPER A ROYALTY-FREE, NON-EXCLUSIVE LICENSE TO INCLUDE
* NOVELL'S SAMPLE CODE IN ITS PRODUCT. NOVELL GRANTS DEVELOPER
* WORLDWIDE DISTRIBUTION RIGHTS TO MARKET, DISTRIBUTE, OR SELL
* NOVELL'S SAMPLE CODE AS A COMPONENT OF DEVELOPER'S PRODUCTS.
* NOVELL SHALL HAVE NO OBLIGATIONS TO DEVELOPER OR
* DEVELOPER'S CUSTOMERS WITH RESPECT TO THIS CODE.
```

```

*
* $name:          CreateNamingContext.java
* $version:       1.0
* $date_modified:Thr, September 11, 2000
* $owner:
* $description:   CreateNamingContext.java demonstrates how to create a
*                 naming context.
*****/
import java.util.*;
import javax.naming.*;
import javax.naming.ldap.*;
import com.sun.jndi.ldap.*;
import com.novell.service.ndssdk.jndi.ldap.ext.*;

public class CreateNamingContext
{
    public static void main(String[] args) {

        if (args.length != 4)
        {
            System.err.println("Usage : java CreateNamingContext <host name>"
                + " <login dn> <password> <partition dn>");
            System.err.println("Example: java CreateNamingContext Acme.com"
                + " cn=admin,o=Acme secret ou=sales,o=Acme");
            System.exit ( -1 );
        }
        try
        {
            int    ldapPort = LdapCtx.DEFAULT_PORT;
            String ldapHost = args[0];
            String loginDN  = args[1];
            String passWord = args[2];
            String namingContextDN = args[3];

            // create a Hashtable object to put environment variables
            Hashtable env = new Hashtable(5, 0.75f);

            env.put(Context.INITIAL_CONTEXT_FACTORY,
                "com.sun.jndi.ldap.LdapCtxFactory");
            env.put(Context.PROVIDER_URL, "ldap://" + ldapHost + ":" + ldapPort);

            env.put(Context.SECURITY_AUTHENTICATION, "simple" );
            env.put(Context.SECURITY_PRINCIPAL, loginDN );
            env.put(Context.SECURITY_CREDENTIALS, passWord );

            // construct a LdapContext object
            LdapContext ctx = new InitialLdapContext(env, null);

            /*
            * Call extended operation to create the naming context. The
            * flag used for creating naming context request is:
            *     LDAP_ENSURE_SERVERS_UP = 1
            */
            CreateNamingContextRequest reqs = new CreateNamingContextRequest(
                namingContextDN,
                NamingContextConstants.LDAP_ENSURE_SERVERS_UP );

            LDAPExtendedResponse resp = (LDAPExtendedResponse)
                ctx.extendedOperation(reqs);
        }
    }
}

```

```

        System.out.println("CreateNamingContext operation succeeded.");
    }

    catch (NamingException e)
    {
        System.err.println("CreateNamingContext operation failed.");
        e.printStackTrace();
    }
    finally
    {
        System.exit(0);
    }
}

```

Here is a second piece of sample code using Novell's LDAP
Extensions and Controls for JNDI:

```

// Sample code file: ListReplicas.java
// Warning: This code has been marked up for HTML

/*****
 * Copyright (c) 2002 Novell, Inc. All Rights Reserved.
 *
 * THIS WORK IS SUBJECT TO U.S. AND INTERNATIONAL COPYRIGHT LAWS
 * AND TREATIES. USE AND REDISTRIBUTION OF THIS WORK IS SUBJECT TO
 * THE LICENSE AGREEMENT ACCOMPANYING THE SOFTWARE
 * DEVELOPMENT KIT (SDK) THAT CONTAINS THIS WORK. PURSUANT TO
 * THE SDK LICENSE AGREEMENT, NOVELL HEREBY GRANTS TO
 * DEVELOPER A ROYALTY-FREE, NON-EXCLUSIVE LICENSE TO INCLUDE
 * NOVELL'S SAMPLE CODE IN ITS PRODUCT. NOVELL GRANTS DEVELOPER
 * WORLDWIDE DISTRIBUTION RIGHTS TO MARKET, DISTRIBUTE, OR SELL
 * NOVELL'S SAMPLE CODE AS A COMPONENT OF DEVELOPER'S PRODUCTS.
 * NOVELL SHALL HAVE NO OBLIGATIONS TO DEVELOPER OR
 * DEVELOPER'S CUSTOMERS WITH RESPECT TO THIS CODE.
 *
 * $name:          ListReplicas.java
 * $version:       1.0
 * $date_modified: Thr, September 11, 2000
 * $owner:
 * $description:   ListReplicas.java is used to show all the replicas in
 *                a server object.
 *****/

import java.util.*;
import java.io.*;
import javax.naming.*;
import javax.naming.ldap.*;
import com.sun.jndi.ldap.*;
import com.novell.service.ndssdk.jndi.ldap.ext.*;

public class ListReplicas
{
    public static void main(String[] args) {

        if (args.length != 4)
        {

```

```

System.err.println("Usage : java GetEffectivePrivileges <host name>"
    + " <login dn> <password> <server dn>");
System.err.println("Example: java GetEffectivePrivileges Acme.com"
    + " cn=admin,o=Acme secret cn=myServer,o=Acme");
System.exit ( -1 );
}
try
{
    int      ldapPort = LdapCtx.DEFAULT_PORT;
    String   ldapHost = args[0];
    String   loginDN  = args[1];
    String   password = args[2];
    String   serverDN = args[3];

    Vector replicas;

    // Create a Hashtable object.
    Hashtable env = new Hashtable(5, 0.75f);

    env.put(Context.INITIAL_CONTEXT_FACTORY,
        "com.sun.jndi.ldap.LdapCtxFactory");
    env.put(Context.PROVIDER_URL, "ldap://" + ldapHost + ":" + ldapPort);
    env.put(Context.SECURITY_AUTHENTICATION, "simple" );
    env.put(Context.SECURITY_PRINCIPAL, loginDN );
    env.put(Context.SECURITY_CREDENTIALS, passWord );

    // Construct a LdapContext object.
    LdapContext ctx = new InitialLdapContext(env, null);

    // Call extended operation to retrieve replica.s
    ListReplicasRequest reqs = new ListReplicasRequest( serverDN );

    ListReplicasResponse resp = (ListReplicasResponse)
        ctx.extendedOperation( reqs );

    // Printout the replica(s) if there is any.
    replicas = resp.getReplicas();

    if ( replicas.isEmpty() )
    {
        System.out.println("\nThe server object '" + serverDN
            + "' has no any replicas:");
    }
    else
    {
        Enumeration enum = replicas.elements();
        System.out.println("\nThe server object '" + serverDN
            + "' has the following replica(s):");
        while ( enum.hasMoreElements() )
            System.out.println( enum.nextElement());
    }

    System.out.println("\nListReplicas operation succeeded.");
}
catch (IOException e)
{
    System.err.println( e.toString() );
    e.printStackTrace();
}
catch (NamingException e)

```

```
{
    System.err.println("ListReplicas operation failed.");
    e.getExplanation();
    e.printStackTrace();
}
finally
{
    System.exit(0);
}
}
```

Reference for Novell LDAP Extensions and Controls for JNDI

To download the Novell LDAP Extensions and Controls for JNDI, applicable documentation and lots more sample code, see: <http://developer.novell.com>.

Self-Check

1. The Directory Tree is made up of?
 - a. objects
 - b. predicates
 - c. binary digits
 - d. LDAP searches
2. What Protocol Stack does LDAP use?
 - a. IPX
 - b. SPX
 - c. TCP/IP
 - d. Netbios
3. Which two describe what eDirectory's database is?
 - a. distributed
 - b. hierarchical
 - c. departmental
 - d. UNIX

4. What technology allows JNDI to be platform independent?
 - a. naming service(s)
 - b. card catalog
 - c. hard-coded for loops
 - d. interface to Beans
5. What does a naming service do?
 - a. locate objects by name
 - b. maintains a set of bindings
 - c. all objects in the naming service are named the same way
 - d. subscribes to the same naming convention
6. What does a context represent?
 - a. a set of interfaces that emit light
 - b. a servlet that enables a leaf tree
 - c. a set of bindings within a naming service that share the same naming convention
 - d. a set of bindings within a naming service that do not share the same naming convention
7. Which are methods of the Context Class?
 - a. bind
 - b. rebind
 - c. unbind
 - d. all of the above
8. JNDI's ability to directly store Java objects allows it to integrate almost transparently into Java applications.
 - a. True
 - b. False

9. Which is a technique for storing Java objects?
- a. serialized data
 - b. stereo data
 - c. binary data
 - d. encrypted data
10. What is another technique(s) for storing Java objects?
- a. reference to a class
 - b. reference to an object
 - c. attributes on a directory leaf
 - d. attributes on a directory context
11. What are the two usage roles for Doc-u-matic?
- a. User and Supervisor
 - b. Administrator and User
 - c. Player and User
 - d. Player and Administrator

-
- Answers:
- 1) a. objects
 - 2) TCP/IP
 - 3) a. distributed
b. hierarchical
 - 4) a. naming service(s)
 - 5) a. locate objects by name
b. maintains a set of bindings
c. all objects in the naming service are named the same way
d. subscribes to the same naming convention
 - 6) c. a set of bindings within a naming service that share the same naming convention
 - 7) d. all of the above
 - 8) a. True
 - 9) a. serialized data
 - 10) b. reference to an object
d. attributes on a directory context
 - 11) b. Administrator and User

Conclusion

Congratulations! You have reached the end of the Utilizing the LDAP Extensions and Controls for JNDI in a Directory Environment course.

Checking for Success

During this course you have learned about the role of JNDI, specifically naming and directories, also how easy it is to deal with multiple platforms running different directories.

You are now familiar enough with Directory and JNDI principles and theory to solve directory problems with JNDI. With the information you've learned you can utilize the resources of a Directory using JNDI in your organization. Feel free to return to this course any time to brush up your skills.

Additional Resources

For more information on JNDI, check out the following resources:

- <http://developer.novell.com/ndk/extjndi.htm>
- <http://java.sun.com/products/jndi/>

To find out about other DeveloperNet University courses that are available to you, visit <http://www.developer.novell.com/education>.

To take advantage of all the other technical knowledge available to you, visit AppNotes online (or find out how you can subscribe to the hard copy publication) at <http://developer.novell.com/research/>.

Copyright © 2002 by Novell, Inc. All rights reserved.
No part of this document may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying and recording, for any purpose without the express written permission of Novell.

All product names mentioned are trademarks of their respective companies or distributors.