

# Java Naming and Directory Service Interfaces

by: Paul J. Perrone and Krishna Chaganti

pperrone@assuredtech.com

chaganti@erols.com

Paul Perrone is the President and a Consultant for Assured Technologies, Inc. in Leesburg, Virginia.

Krishna Chaganti is a Consultant in Reston, Virginia.

---

Featured in *Distributed Computing*, October 1999; vol 2, issue 9; pp 12-16,20.

---

A naming service is the principal mechanism used in distributed enterprise systems for obtaining handles to objects via the use of a name identifying that object. Directory services extend the capabilities of naming services and associate a set of attributes with each registered object. Directory service clients can search for directory objects using search criteria expressed in terms of a directory object's attributes. The Java Naming and Directory Interface (JNDI) provides a standard way for Java applications to interface with a variety of naming and directory services. JNDI provides the benefit of allowing programmers and Java applications a single interface model for seamless and interchangeable interaction with various naming and directory services.

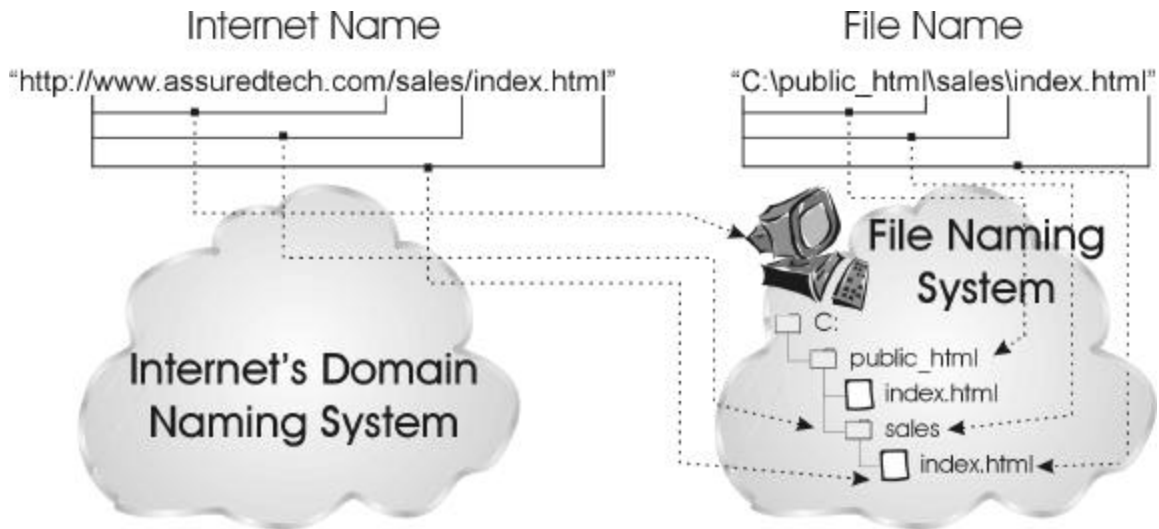
## Naming Services Overview

An object in a system corresponds to an instance in memory or some range of addresses relating to some discrete collection of functions and/or data. "Handles" to such objects are often referred to as references or pointers. Such handles can be associated with objects residing in a process local to your current execution environment or with distributed objects on an entirely different machine. However, these handles are often not human readable and frequently have communications protocol, platform, and current execution environment dependencies.

A name is simply a human-readable logical value for referring to an object in a system. A name can be for example:

- A file name referring to a file object reference in a file naming system.
- An Internet host name referring to a network element's IP address in a DNS naming system.
- A management service name referring to a CORBA server object on a remote machine in a CORBA naming system.

Each name is expressed using a syntax convention understood by the naming system for which it is relevant. The name's meaning is always specific to some context. Thus, as illustrated in Figure 1, while the name "index.html" on your computer hosting a web server refers to a specific file that you have created in the directory "C:\public\_html", this same filename in your web server's directory "C:\public\_html\sales" will refer to an entirely different file object. The name "index.html" has to be put in context to know what specific file object is being referenced. Such file name to file object mapping is provided by your machine's file naming system. Web clients can access these same files by first consulting the Domain Name Service to locate your host machine and then are subsequently routed by your web server to the desired file relative to the web server's root context.



**Figure 1: Naming Systems Example**

Each naming context thus provides a set of name to object bindings whereby the object may be an actual system resource (e.g. file object) or perhaps another context (i.e. a sub-context). Each context can have its own standard for naming the objects and sub-contexts to which it binds names. However, a naming system provides a standard for managing a set of contexts with the same naming convention and for providing a standard means for binding names to objects and resolving objects to names.

## Directory Services Overview

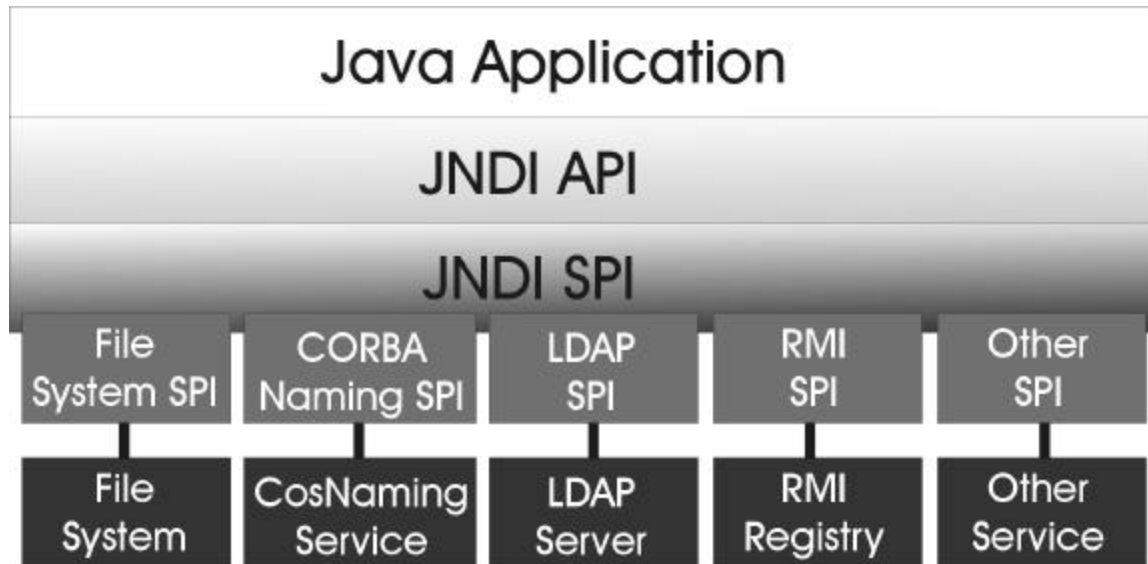
A directory service can be viewed as a type of naming service which is augmented with the capability to perform searches for objects based on search filter criteria. Directory services also allow for the modification of object attributes in the directory service tree. The Object Management Group (OMG) provides a standard classification of naming services as those services providing pure name to object associations akin to a telephone white pages. Directory services may indeed provide naming services but their additional services for retrieval and advertisement by object characteristics render them more like telephone yellow pages. Directory services thus are similar in concept to the OMG's form of yellow page service known as the CORBA Trading Service.

Directory services typically have a hierarchical context structure with each contained directory object corresponding to some enterprise resource such as a file, a printer, or a user's profile. This directory object itself is actually manifested in a directory service as a collection of attributes describing the resource and perhaps containing distributed object references and serialized objects. A directory object in a directory service tree can be retrieved by its name or via use of search criteria expressed as a function of the directory object's name and attributes. When using directory services to add, delete, modify, and search for attributes of directory objects, they bear a striking resemblance to more traditional database services. However, directory services are by and large used for data retrieval versus updates and typically lack the transaction semantics and large-scale data warehouse support that databases offer.

## Java Naming and Directory Interface (JNDI)

The Java Naming and Directory Interface (JNDI) provides a standard way for Java applications to interface with different naming services and directory services (see Figure 2). JNDI provides a Java API for commonly interfacing with any naming or directory service for which there exists an adapter to map JNDI API calls to the specific calls provided by the particular naming or directory service provider. After a few initialization parameters are set which tell the JNDI API what particular service to delegate calls, all calls to the JNDI API are properly delegated to the particular service via the adapter. This adapter is referred to as a

Service Provider Interface (SPI). SPIs are provided by Sun Microsystems for some of the most popular services, by the vendors of services wishing to provide their users a JNDI interface, and by third-parties.



**Figure 2: JNDI Architecture**

When using JNDI, the first step (aside from having a Java Runtime Environment) is to obtain the JNDI class libraries. These libraries can be downloaded from the JavaSoft web site at <http://www.javasoft.com/products/jndi>. As of the writing of this paper, the latest version of JNDI available is version 1.2 (not to be confused with or tied to a JDK version).

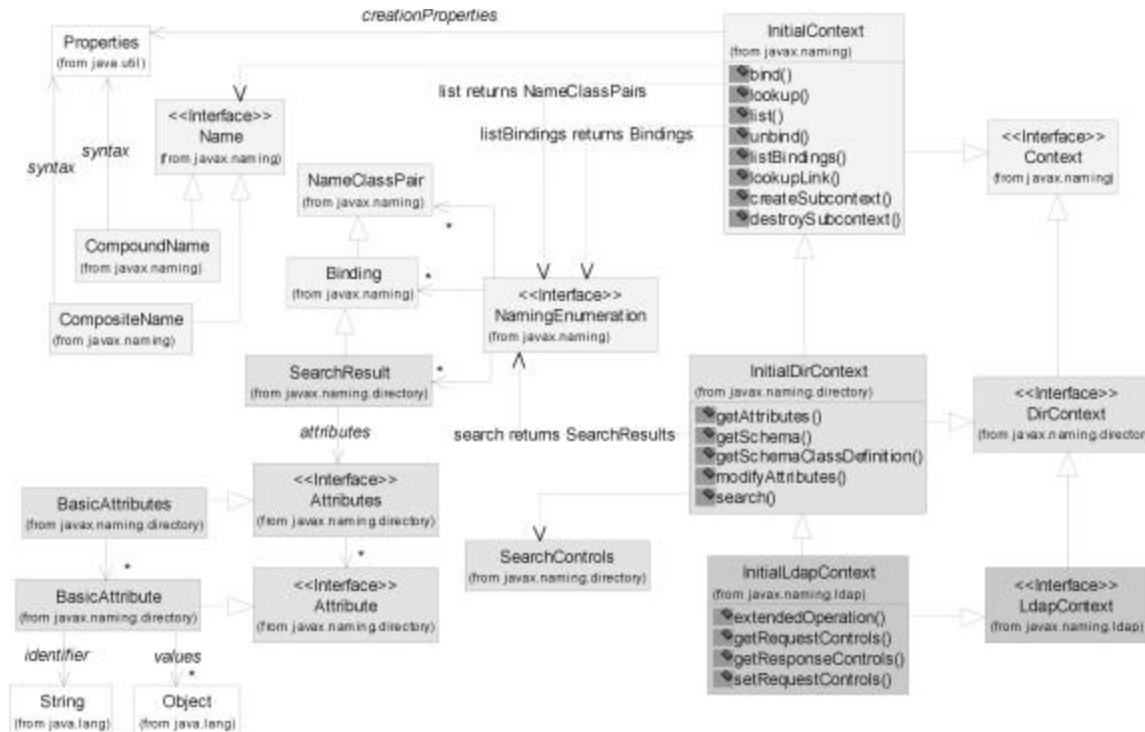
You will also need an SPI for the naming or directory system used in your Java application. The JavaSoft web site also offers a collection of links to some of the more popular naming and directory system SPIs at <http://www.javasoft.com/products/jndi/serviceproviders.html>. Vendor-specific SPI implementations are also typically shipped with the vendor's product and may also include the specific JNDI class library version for which it has been certified for use. Of course the availability of the actual naming and directory service itself is specific to each type of naming and directory system. The file naming system, for example, is a given for most systems, but a naming service for CORBA will typically be purchased off the shelf or packaged as part of an overall vendor-specific product offering.

The JNDI class libraries are partitioned into the following packages:

- *javax.naming* – Contains all of the core JNDI API classes and interfaces used by Java applications wishing access to a variety of naming services.
- *javax.naming.directory* – Contains the JNDI API classes augmented for performing more sophisticated directory service functionality extending the basic naming service functionality.
- *javax.naming.ldap* – Contains the JNDI API classes and interfaces (new as of JNDI v1.2) for specific support of the more advanced management features of the LDAPv3 standard when using the LDAP SPI.
- *javax.naming.event* – Contains the JNDI API classes and interfaces (new as of JNDI v1.2) for providing naming and directory system event notification services.
- *javax.naming.spi* – Contains the JNDI SPI classes and interfaces for implementers of SPIs to map standard JNDI API calls to calls specific to a particular naming or directory service.

The JNDI client API user will first be required to create an initial context to the naming or directory service of interest. This initial context establishes a connection with the naming or directory service when constructed with a set of properties describing the specific naming or directory service provider library to

use, the URL of the naming or directory service process, and perhaps a user name and credentials. Other properties may also be set and are defined in the *javax.naming.Context* interface. As can be seen from Figure 3, three main interfaces and three main classes are used to accomplish this task. An *InitialContext* class can be used for naming services. An *InitialDirContext* class can be used for directory services with a specialized *InitialLdapContext* class to be used for LDAPv3-style extended directory service operations.



**Figure 3: JNDI API Classes and Interfaces**

Once an initial context to a naming or directory service is established, the root context or sub-context reference may be used to perform various operations on the context depending on the type of context. Naming contexts can be used to bind and unbind names and objects, list names and name bindings, and lookup object references of named objects. A *Name* interface and two classes realizing this interface (*CompoundName* and *CompositeName*) exist to help manipulate names in a fashion which is sometimes easier than using string names. Names have syntax properties which have different values for different naming and directory services. Compound names are sequences of simple atomic name components. Composite names are more complex and allow for the representation of names from multiple name spaces. These name spaces may even span different naming and directory service systems offering the API user a seamless interface to a heterogeneous distributed enterprise system.

When listing the names and class names of bound objects in a context, a collection (*NamingEnumeration*) of object name and class name pairs (*NameClassPair*) will be returned to the API caller. The names and their bound objects in a context can also be listed and returned as a collection of *Binding* objects. The directory service context extends the basic naming context with operations for searching for objects and attributes as well as modifying the attributes and viewing the schema structure of directory objects. The API user creates a set of search controls (*SearchControl*) to describe the search criteria as a function of directory object attributes. Searching yields a collection of search results (*SearchResult*) whereby each search result is simply a collection of attributes (*Attributes*). Each attribute (*Attribute*) has an identifier and a set of values. The specialized LDAP context exists to support those operations useful for the LDAP v3 standard not supported by the more generic directory service context.

## Naming Service Provider Examples

### Naming Files

The file naming system is the most basic and common naming system that developers encounter. In a file naming system, file names correspond to naming service names, file system directories correspond to the context which contains names, and the file objects and descriptors correspond to the system resource handles of interest. Different file systems will have different naming syntax conventions. Thus, while Windows-based file systems use the backslash “\” to separate context-space and components of a name, Unix-based systems use the forward slash “/”. A file system JNDI SPI exists for use by the JNDI API and is freely downloadable from the JavaSoft web site.

### RMI Naming

RMI is Java’s built-in means for providing Java applications with an easy way to create distributed Java applications. The JDK provides a *java.rmi* API package as well as a registry service for distributed Java RMI servers to register their services. The *java.rmi.Naming* class provides a means for binding names to objects, listing object names, looking up objects given a name, and unbinding names from objects. A JNDI SPI also exists for RMI which enables applications to interface with the RMI naming service through the JNDI API.

### CORBA Naming

The OMG’s CORBA Object Naming Service presents the primary and standard way for mapping between names and objects on a CORBA ORB. The Object Naming Service was proposed by the OMG as a means to provide a standard interface to a variety of underlying naming services. The idea was that the Object Naming Service standard would serve as the language-independent and standard way to wrap existing name servers for connectivity from a variety of clients. As you are now aware, this goal for providing a standard way to interface with naming services is also the goal of JNDI but only in the context of Java clients. A CORBA naming service (a.k.a. CosNaming) SPI is available for download on JavaSoft web site.

### DNS

The Domain Name Service (DNS) provides for a translation of the hierarchically-defined machine host names that we are all familiar with (e.g. www.yahoo.com) to and from IP addresses (e.g. 204.71.200.68). These IP addresses are used by network routing elements to determine how to deliver IP-based messages to a host machine. By using human-readable and structured names, the task of the developer is simplified and less dependent on a priori knowledge of the internals of routing tables and host machine configurations. At the time of this paper’s writing, no known JNDI SPI for the Domain Name Service (DNS) exists.

## Directory Service Provider Examples

### NIS

The Network Information System (NIS) is a directory service (formerly referred to as the Yellow Pages or “YP”) developed by Sun Microsystems and widely used in Unix-based platforms. NIS+ is a later version of NIS providing enhancements to security and other services. NIS/NIS+ provides clients with access to files and other system resources on local area networks. A JNDI SPI exists for NIS and is downloadable from the JavaSoft web site.

### NDS

The Novell Directory Service (NDS) is a popular directory service for managing enterprise network resources and network user information. NDS’ multi-platform support and its easy to use GUI administration toolkit makes it a popular choice for use by network administrators. Novell currently offers a JNDI SPI for Java applications to interface with this popular directory service.

## **LDAP**

Lightweight Directory Access Protocol (LDAP) services are used to store information such as user, organizational, file, and network resource information. LDAP is a lighter weight version of the X.500's Directory Access Protocol (DAP) standard and offers efficiencies by being tailored for information retrieval versus updates and running over the TCP/IP protocol. The LDAP context structure begins with a root context which is then followed by a country sub-context, followed by an organizational sub-context, which in turn is followed by an individual/resource sub-context (i.e. people or computing resource). Not only is there a JNDI SPI for LDAP, but there also exists a special *javax.directory.ldap* package for supporting some of the more sophisticated LDAP v3 features.

## **Conclusions**

Naming and directory services are fundamental components of any distributed enterprise system. Various naming and directory service interfaces can be abstracted by the JNDI API for Java clients of those services. JNDI offers both the developer and Java applications a stable interface for interchanging and seamlessly communicating with a variety of naming and directory service providers. Many common service providers already have access support through the JNDI API and more SPIs are on the way. Enterprise projects will continue to use the JNDI API as a fundamental component when building distributed enterprise Java applications.