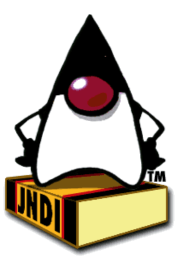
JNDI Java Naming & Directory Interface™

DEC 1997

*Java Naming and Directory Interface™ (**JNDI**) is a new addition to JavaSoft's platform APIs. It provides Java applications a unified interface to multiple naming and directory services in the enterprise. As part of the Java Enterprise API set,* **JNDI** *enables seamless connectivity to heterogeneous enterprise naming and directory services. Java developers can now build powerful and portable directory-enabled applications using this industry-standard interface.*

The **JNDI** *specification was developed by JavaSoft with a number of leading industry partners, including SunSoft, Novell, Netscape, IBM and HP.*

The beta version of the specification and code are now available for download at the JavaSoft web site.



Technical Overview

Directory services play a vital role in Intranets and Internets by providing access to a variety of information about users, machines, networks, services, and applications. By its very nature, a directory service incorporates a naming facility for providing human understandable namespaces that characterize the arrangement and identification of the various entities.

The computing environment of an enterprise typically consists of several naming facilities often representing different parts of a *composite* namespace. For example, the Internet Domain Name System (DNS) may be used as the top-level naming facility for different organizations within an enterprise. The organizations themselves may use a directory service such as LDAP or NDS or NIS. From a user's perspective, there is one namespace consisting of composite names. URLs are examples of composite names because they span namespaces of multiple naming facilities. Applications which use directory services must support this user perspective.

Many Java application developers can benefit from a directory service API that is not only independent of the particular directory or naming service implementation, but also enables seamless access to directory objects through multiple naming facilities. In fact, an application can attach its own objects to the namespace. Such a facility enables any Java application to discover and retrieve Java objects of any type.

JNDI is an API specified in Java™ that provides directory and naming functionality to applications written in Java. It is defined to be independent of any specific directory service implementation. Thus, a variety of directories, new and existing ones in the installed base, can be accessed in a common way.

Directory service developers can benefit from a service-provider capability that enables them to incorporate their respective implementations without requiring changes to the client.

JNDI also defines a service provider's interface which allows various directory and naming service drivers to be plugged in.

Examples

Here are two examples to briefly illustrate some of the more commonly used features of **JNDI**.

An application that wants to access a printer needs the corresponding printer object. This is simply done as follows:

```
prt = (Printer)
 building7.lookup("puffin");
prt.print(document);
```

where `building7` is the naming context representing a physical building that provides a convenient context for referring to the printers.

JNDI does all the work of locating the information needed to construct the printer object.

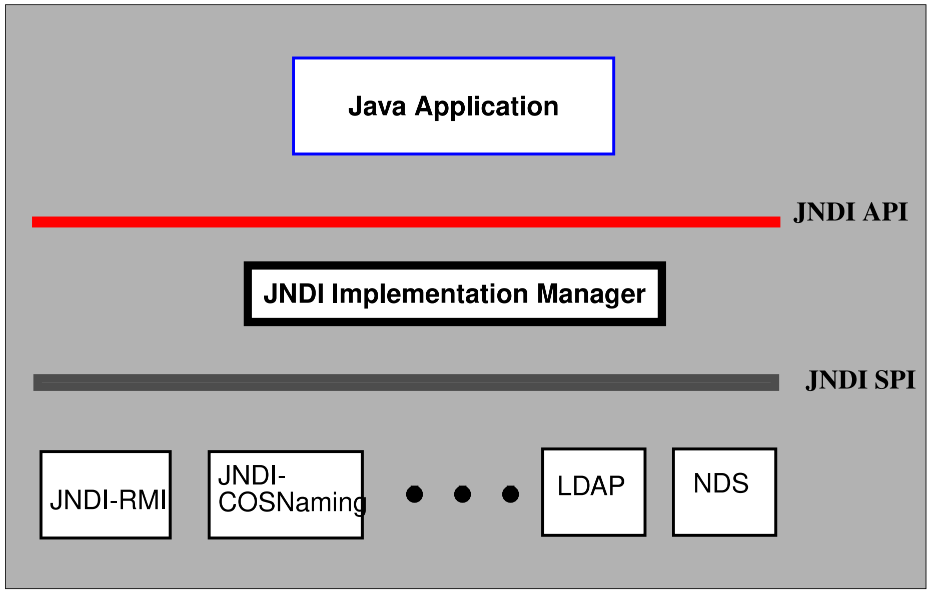
As another example, an application that wants to find a person's phone numbers, which are stored in the organization's directory, can simply do:

```
String[] attrs = {"workPhone",
 "cellPhone", "faxNumber"};
bobsPhones =
  directory.getAttributes(
  "cn=Bob,o=Widget,c=US",
  attrs);
```

If there may be several Bobs in the Widget organization, the application can search the organization's directory to find the right Bob as follows:

```
bob = directory.search(
  "o=Widget,c=US","cn=Bob",
  controls);
```

Other application examples include access to security credentials stored in an enterprise-wide directory service, access to electronic mail addresses, and access to addresses of a variety of existing services such as databases, network file systems, etc.



Overview of Interfaces

The **JNDI** API is contained in two packages: `javax.naming` for the naming operations, and `javax.naming.directory` for directory operations. The **JNDI** service provider interface is contained in the package `javax.naming.spi`.

The Naming Interface — `javax.naming`

`javax.naming.Context` is the core interface that specifies a naming context. It defines basic operations such as adding a name-to-object binding, looking up the object bound to a specified name, listing the bindings, removing a name-to-object binding, creating and destroying subcontexts of the same type, *etc.*

`Context.lookup()` is the most commonly used operation. The context implementation can return an object of whatever class is required by the Java client. For example, a client might use the name of a printer to look up the corresponding `Printer` object, and then print to it directly:

```
Printer printer = (Printer)
 ctx.lookup("treekiller");
printer.print(report);
```

The client is not exposed to any naming service implementation. In fact, a new type of naming service can be introduced without requiring the application to be modified or even disrupted if it is running.

The Directory Interface — `javax.naming.directory`

Directory Objects and Attributes. The `DirContext` interface enables the directory capability by defining methods for examining and updating attributes associated with a directory object. Each directory object contains a set of zero or more objects of class `Attribute`. Each attribute is denoted by a string identifier and can have zero or more values of any type.

Directory Objects as Naming Context. The `DirContext` interface also behaves as a naming context by extending the `Context` interface. This means that any directory object can also provide a naming context. In addition to a directory object keeping a variety of information about a person, for example, it is also a natural naming context for resources associated with that person: a person's printers, file system, calendar, *etc.*

Searches. The `DirContext` interface supports content-based searching of directories. In the simplest and most common form of usage, the application specifies a set of attributes — possibly with specific values — to match. It then invokes the `DirContext.search()` method on the directory object, which returns the matching directory objects along with the requested attributes.

The Service Provider Interface — `javax.naming.spi`

The **JNDI** SPI provides the means by which different naming/directory service providers can develop and hook up their implementations so that the corresponding services are accessible from applications that use **JNDI**. In addition, because **JNDI** allows specification of names that span multiple namespaces, if one service provider implementation needs to interact with another in order to complete an operation, the SPI provides methods that allow different provider implementations to cooperate to complete client **JNDI** operations.

For More Information

See *http://java.sun.com/products/jndi/.*