Service Location Working Group
INTERNET DRAFT

James Kempf
Jonathan Wood
Sun Microsystems, Inc
24 June 1998

The jndi-drivers Abstract Service Type
draft-ietf-srvloc-jndi-drivers-00.txt

Status of This Memo

Abstract

This document describes the jndi-drivers abstract type. The jndi-drivers service provides access to drivers (also known as service provider classes) for the Java Naming and Directory Interface (JNDI). This type can be used in conjunction with the Service Location Protocol.

1. Introduction

The Service Location Protocol is defined in [1]. Service templates and abstract types are defined in [2]. JNDI is defined in [3]. The

jndi-drivers abstract type is intended to organize information pertaining to the location and access method needed to obtain dynamically JNDI drivers.

Java class files can be obtained off the local file system or from the network. Hence drivers for JNDI need not be bundled with an application which uses JNDI; they can be downloaded from the network and dynamically instantiated (for instance, by a Java class-loader). The jndi-drivers type contains all information necessary to complete this process.

The concrete type should be the access protocol used to retrieve the driver. The URL should also specify the host which provides access to the driver, and a path relative to the server host which names the location of the driver's class file archive.

2. Example Scenario

This example demonstrates how to use this service type in conjunction with SLP and JNDI to dynamically instantiate a service provider for LDAP. LDAP is defined in [4].

An application running JNDI has only the core JNDI classes available to it locally. The application knows that it wishes to talk to an LDAP server, and has the information it needs to talk to that LDAP server: the server's name is "andromeda", serving the base suffix "o=sun,c=us", and it listens on port 389. (The application need not be preconfigured with the location of the LDAP service in order to make use of the dynamic loading of a JNDI driver for LDAP. The application could use SLP [1] to discover the LDAP server, though further discussion of this is beyond the scope of this document.)

The application does not have the LDAP service provider classes, nor does it know where to obtain the classes. It does have the ability to use SLP, and can retrieve class files using the http protocol.

In order to discover the location of the LDAP service provider classes, the application issues a SLP service request for the service type "service:jndi-drivers" with the predicate string "(&(service-type=ldap)(access-protocol=http))".

The LDAP service provider classes are offered by the http server "jserver.sun.com", which is also advertising this offering via SLP. "jserver" thus sends a SLP service reply to the requesting application with the following URL:


```
service:jndi-drivers:http://jserver.sun.com/java/drivers/ldap.jar;\
   class=com.sun.jndi.LdapCtx;\
   factory=com.sun.jndi.LdapCtxFactory
```

The application now has enough information to retrieve the class files it needs. It will use the JNDI NamingManager to create an LDAP service provider context; to do so, it must first decompose the service: URL into a JNDI Reference. The following Java code fragment shows how this is done. Note that the ServiceURL object is defined in [5], which also defines an API for using SLP through which ServiceURLs can be obtained. Error handling is omitted for brevity.

```
 ServiceURL surl = // result from service request

 // Compose a JNDI Reference from the service: URL
 // Get the class name
 String urlPart = surl.getURLPath();
 String className = urlPart.substring(urlPart.indexOf("class="));
 className = className.substring(className.indexOf("=") + 1,
                                 className.indexOf(";"));

 // Get the factory class name
 String fact = urlPart.substring(urlPart.indexOf("factory="));
 fact = fact.substring(fact.indexOf("=") + 1);

 // Reconstruct the URL for the class files
 String type = surl.getServiceType();
 String proto = type.getConcreteTypeName();
 String host = surl.getHost();
 String factLoc = proto + "://" + host +
                  urlPart.substring(0, urlPart.indexOf(";"));

 // Compose a JNDI RefAddr which will bootstrap the driver
 StringRefAddr refAddr = new StringRefAddr("URL",
                           "ldap://andromeda:389/o=sun,c=us");
```

```
    // Compose a JNDI Reference with all information needed to
    // instantiate and bootstrap the provider
    Reference ref = new Reference(className, refAddr, fact, factLoc);

    // Install a permissive security manager here ...
    System.setSecurityManager(new PermissiveSecurityManager());

    // Retrieve, instantiate, and bootstrap the provider
    DirContext ctx = (DirContext)
                      NamingManager.getObjectInstance(ref,
                                                      null, null, null);
```


The application now has a working LDAP DirContext object.

2. The "jndi-drivers" Abstract Service

```
  -------------------------template begins here-----------------------

type = jndi-drivers

version = 0.1

language = en

scheme-description =
The jndi-drivers service provides the following information:
1) The name of a service class,
2) The name of a factory object to create the service class,
3) A URL locating the factory object and service object Java code.
This is an abstract type. The concrete type should be the access
protocol used to retrieve the driver.
This template takes advantage of the abstract types introduced
with SLP V2 [1].

url-syntax =
url-part = url ";class=" class name ";factory=" factory-class-name
url = an URL as defined in [6]
class-name = ;Fully qualified Java name of service class
factory-class-name = ;Fully qualified Java name of the factory object's class

service-type = STRING L
```

```
#The SLP service type name for the driver.

jndi-driver-version = STRING L
#The driver's version number.

access-protocol = string
# The protocol used to access this driver. This attribute
# is useful for SLP UAs which wish to query for a set of preferred
# access protocols. This attribute should match the access protocol
# given in the concrete type.

  ---------------------------template ends here-----------------------

contacts = "James Kempf" <james.kempf@eng.sun.com>
           "Jonathan Wood" <jonathan.wood@eng.sun.com>

security-considerations = Drivers obtained from these service: URLs
  will ultimately be executed within process space of the
  instantiating application. As such, the application should take
```



```
  care to ensure that the driver is valid, and not malicious. Two
  possible means of ensuring a driver's validity are the use of SLP
  protected scopes, or by utilizing signed Java Archives (JAR files)
  provided by Sun's Java Development Kit (JDK) [7]
```

References:

[1] E. Guttman, C. Perkins, J. Veizades, M. Day, Service Location Protocol. draft-ietf-svrloc-protocol-v2-06.txt, May 1998 (work in progress)

[2] E. Guttman, C. Perkins, J. Kempf, Service Templates and service: Schemes. draft-ietf-svrloc-service-scheme-10.txt March, 1998 (work in progress)

[3] The Java Naming and Directory Interface (TM) Specification, Sun Microsystems, Inc. Feb 1998. http://java.sun.com/jndi/.

[4] M. Wahl, T. Howes, S. Kille, The Lightweight Directory Access Protocol (v3). RFC 2251

[5] J. Kempf, E. Guttman, An API for Service Location

draft-ietf-svrloc-api-05.txt, May 1998 (work in progress)

[6] T. Berners-Lee, R. Fielding, and L. Masinter. Uniform Resource Locators (URL): Generic Syntax and Semantics. RFC1738 as amended by RFC1808

[7] The Java Development Kit
Sun Microsystems, Inc. http://java.sun.com/products/jdk

Authors' Addresses

James Kempf
Sun Microsystems
901 San Antonio Avenue
Palo Alto, CA 94043
USA
Phone: +1 650 786-5890
email: James.Kempf@Eng.Sun.COM

Jonathan Wood
Sun Microsystems
901 San Antonio Avenue
Palo Alto, CA 94043
USA
Phone: +1 650 786-4815
email: Jonathan.Wood@Eng.Sun.COM