## NAME

tnameserv − Interface Definition Language (IDL).

## SYNOPSIS

**tnameserve −ORBInitialPort** [ *nameserverport* ]

-ORBInitialPort *nameserverport*

The initial port where the naming service listens for the bootstrap protocol used to implement the ORB **resolve_initial_references** and **list_initial_references** methods.

## DESCRIPTION

Java IDL includes the Object Request Broker Daemon (ORBD). ORBD is a daemon process that contains a Bootstrap Service, a Transient Naming Service, a Persistent Naming Service, and a Server Manager. The Java IDL tutorials all use ORBD, but you can substitute the **tnameserv** command for the **orbd** command in any of the examples that use a Transient Naming Service.

See orbd(1) or Naming Service at
http://docs.oracle.com/javase/8/docs/technotes/guides/idl/jidlNaming.html

The CORBA Common Object Services (COS) Naming Service provides a tree-structure directory for object references similar to a file system that provides a directory structure for files. The Transient Naming Service provided with Java IDL, **tnameserv**, is a simple implementation of the COS Naming Service specification.

Object references are stored in the name space by name and each object reference-name pair is called a name binding. Name bindings can be organized under naming contexts. Naming contexts are name bindings and serve the same organizational function as a file system subdirectory. All bindings are stored under the initial naming context. The initial naming context is the only persistent binding in the name space. The rest of the name space is lost when the Java IDL naming service process stops and restarts.

For an applet or application to use COS naming, its ORB must know the port of a host running a naming service or have access to an initial naming context string for that naming service. The naming service can either be the Java IDL naming service or another COS-compliant naming service.

### START THE NAMING SERVICE

You must start the Java IDL naming service before an application or applet that uses its naming service. Installation of the Java IDL product creates a script (Oracle Solaris: **tnameserv**) or executable file (Windows: **tnameserv.exe**) that starts the Java IDL naming service. Start the naming service so it runs in the background.

If you do not specify otherwise, then the Java IDL naming service listens on port 900 for the bootstrap protocol used to implement the ORB **resolve_initial_references** and **list_initial_references methods**, as follows:

**tnameserv −ORBInitialPort nameserverport&**

If you do not specify the name server port, then port 900 is used by default. When running Oracle Solaris software, you must become the root user to start a process on a port below 1024. For this reason, it is recommended that you use a port number greater than or equal to 1024. To specify a different port, for example, 1050, and to run the naming service in the background, from a UNIX command shell, enter:

**tnameserv −ORBInitialPort 1050&**

From an MS-DOS system prompt (Windows), enter:

**start tnameserv −ORBInitialPort 1050**

Clients of the name server must be made aware of the new port number. Do this by setting the **org.omg.CORBA.ORBInitialPort** property to the new port number when you create the ORB object.

### RUN THE SERVER AND CLIENT ON DIFFERENT HOSTS

In most of the Java IDL and RMI-IIOP tutorials, the naming service, server, and client are all running on the development machine. In real-world deployment, the client and server probably run on different host machines from the Naming Service.

For the client and server to find the Naming Service, they must be made aware of the port number and host on which the naming service is running. Do this by setting the **org.omg.CORBA.ORBInitialPort** and **org.omg.CORBA.ORBInitialHost** properties in the client and server files to the machine name and port number on which the Naming Service is running. An example of this is shown in Getting Started Using RMI-IIOP at http://docs.oracle.com/javase/8/docs/technotes/guides/rmi-iiop/rmiiiopexample.html

You could also use the command-line options **-ORBInitialPort nameserverport#** and **-ORBInitialHost nameserverhostname** to tell the client and server where to find the naming service. For one example of doing this using the command-line option, see Java IDL: The Hello World Example on Two Machines at http://docs.oracle.com/javase/8/docs/technotes/guides/idl/tutorial/jidl2machines.html

For example, suppose the Transient Naming Service, **tnameserv** is running on port 1050 on host **nameserverhost**. The client is running on host **clienthost,** and the server is running on host **serverhost**.

Start **tnameserv** on the host **nameserverhost**:

**tnameserv −ORBInitialPort 1050**

Start the server on the **serverhost**:

**java Server −ORBInitialPort 1050 −ORBInitialHost nameserverhost**

Start the client on the **clienthost**:

**java Client −ORBInitialPort 1050 −ORBInitialHost nameserverhost**

### STOP THE NAMING SERVICE

To stop the Java IDL naming service, use the relevant operating system command, such as **kill** for a Unix process or **Ctrl+C** for a Windows process. The naming service continues to wait for invocations until it is explicitly shut down. Note that names registered with the Java IDL naming service disappear when the service is terminated.

## OPTIONS

-J*option*

Passes **option** to the Java Virtual Machine, where **option** is one of the options described on the reference page for the Java application launcher. For example, **-J-Xms48m** sets the startup memory to 48 MB. See java(1).

## EXAMPLES

### ADD OBJECTS TO THE NAME SPACE

The following example shows how to add names to the name space. It is a self-contained Transient Naming Service client that creates the following simple tree.

**Initial Naming Context**
**plans**
**Personal**
**calendar**
**schedule**

In this example, **plans** is an object reference and **Personal** is a naming context that contains two object references: **calendar** and **schedule**.

**import java.util.Properties;**

```
import org.omg.CORBA.*;
import org.omg.CosNaming.*;
public class NameClient {
   public static void main(String args[]) {
      try {
```

In Start the Naming Service, the **nameserver** was started on port 1050. The following code ensures that the client program is aware of this port number.

```
Properties props = new Properties();
props.put("org.omg.CORBA.ORBInitialPort", "1050");
ORB orb = ORB.init(args, props);
```

This code obtains the initial naming context and assigns it to **ctx**. The second line copies **ctx** into a dummy object reference **objref** that is attached to various names and added into the name space.

```
NamingContext ctx =
   NamingContextHelper.narrow(
      orb.resolve_initial_references("NameService"));
 NamingContext objref = ctx;
```

This code creates a name **plans** of type **text** and binds it to the dummy object reference. **plans** is then added under the initial naming context using the **rebind** method. The **rebind** method enables you to run this program over and over again without getting the exceptions from using the **bind** method.

```
NameComponent nc1 = new NameComponent("plans", "text");
NameComponent[] name1 = {nc1};
ctx.rebind(name1, objref);
System.out.println("plans rebind successful!");
```

This code creates a naming context called **Personal** of type **directory**. The resulting object reference, **ctx2**, is bound to the **name** and added under the initial naming context.

```
NameComponent nc2 = new NameComponent("Personal", "directory");
NameComponent[] name2 = {nc2};
NamingContext ctx2 = ctx.bind_new_context(name2);
System.out.println("new naming context added..");
```

The remainder of the code binds the dummy object reference using the names **schedule** and **calendar** under the **Personal** naming context (**ctx2**).

```
NameComponent nc3 = new NameComponent("schedule", "text");
NameComponent[] name3 = {nc3};
ctx2.rebind(name3, objref);
System.out.println("schedule rebind successful!");
NameComponent nc4 = new NameComponent("calender", "text");
NameComponent[] name4 = {nc4};
ctx2.rebind(name4, objref);
System.out.println("calender rebind successful!");
      } catch (Exception e) {
         e.printStackTrace(System.err);
      }
   }
}
```

## BROWSING THE NAME SPACE

The following sample program shoes how to browse the name space.

```
import java.util.Properties;
import org.omg.CORBA.*;
import org.omg.CosNaming.*;
public class NameClientList {
   public static void main(String args[]) {
      try {
```

In Start the Naming Service, the **nameserver** was started on port 1050. The following code ensures that the client program is aware of this port number.

```
Properties props = new Properties();
props.put("org.omg.CORBA.ORBInitialPort", "1050");
ORB orb = ORB.init(args, props);
```

The following code obtains the initial naming context.

```
NamingContext nc =
NamingContextHelper.narrow(
   orb.resolve_initial_references("NameService"));
```

The **list** method lists the bindings in the naming context. In this case, up to 1000 bindings from the initial naming context will be returned in the **BindingListHolder**; any remaining bindings are returned in the **BindingIteratorHolder**.

```
BindingListHolder bl = new BindingListHolder();
BindingIteratorHolder blIt= new BindingIteratorHolder();
nc.list(1000, bl, blIt);
```

This code gets the array of bindings out of the returned **BindingListHolder**. If there are no bindings, then the program ends.

```
Binding bindings[] = bl.value;
if (bindings.length == 0) return;
```

The remainder of the code loops through the bindings and prints outs the names.

```
for (int i=0; i < bindings.length; i++) {
   // get the object reference for each binding
   org.omg.CORBA.Object obj = nc.resolve(bindings[i].binding_name);
   String objStr = orb.object_to_string(obj);
   int lastIx = bindings[i].binding_name.length−1;
   // check to see if this is a naming context
   if (bindings[i].binding_type == BindingType.ncontext) {
      System.out.println("Context: " +
         bindings[i].binding_name[lastIx].id);
   } else {
      System.out.println("Object: " +
         bindings[i].binding_name[lastIx].id);
   }
}
} catch (Exception e) {
```

```
            e.printStackTrace(System.err)
        }
    }
}
```

## SEE ALSO

- orbd(1)