BEA WebLogic Enterprise

Using the CORBA Name Service

WebLogic Enterprise 5.1
Document Edition 5.1
May 2000
Using the CORBA Name Service

<table>
<thead>
<tr>
<th>Document Edition</th>
<th>Date</th>
<th>Software Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1</td>
<td>May 2000</td>
<td>BEA WebLogic Enterprise 5.1</td>
</tr>
</tbody>
</table>
Contents

About This Document

What You Need to Know ................................................................................... vii
e-docs Web Site ................................................................................................ viii
How to Print the Document ........................................................................... viii
Related Information ........................................................................................ viii
Contact Us! ...................................................................................................... ix
Documentation Conventions .......................................................................... ix

1. Overview of the CORBA Name Service

The CORBA Name Service ............................................................................ 1-1
Understanding the CORBA Name Service ............................................... 1-3

2. CORBA Name Service Reference

CORBA Name Service Commands ................................................................. 2-1
  cns ....................................................................................................... 2-3
  cnbind ............................................................................................ 2-6
  cnsls .............................................................................................. 2-9
  cnunbind ....................................................................................... 2-11
Capabilities and Limitations of the CORBA Name Service ...................... 2-12
Getting the Initial Reference to the NameService Environmental Object ... 2-13
The CosNaming Data Structures Used by the CORBA Name Service ...... 2-13
The NamingContext Object ........................................................................ 2-14
  CosNaming::NamingContext::bind() .................................................. 2-15
  CosNaming::NamingContext::bind_context() .................................. 2-16
  CosNaming::NamingContext::bind_new_context() ............................ 2-17
  CosNaming::NamingContext::destroy() .......................................... 2-18
  CosNaming::NamingContext::list() .................................................. 2-19
CosNaming::NamingContext::new_context() ........................................ 2-20
CosNaming::NamingContext::rebind() ............................................. 2-21
CosNaming::NamingContext::rebind_context() ............................... 2-22
CosNaming::NamingContext::resolve .............................. 2-23
CosNaming::NamingContext::unbind............................................... 2-24

The NamingContextExt Object ............................................................................. 2-24
CosNaming::NamingContextExt::resolve_str() ................................ 2-26
CosNaming::NamingContextExt::to_name().................................... 2-27
CosNaming::NamingContextExt::to_string() ................................... 2-28
CosNaming::NamingContextExt::to_URL() .................................... 2-29

The BindingIterator Object.............................................................................. 2-29
CosNaming::BindingIterator::destroy() ............................................ 2-31
CosNaming::BindingIterator::next_n() ............................................. 2-32
CosNaming::BindingIterator::next_one() ......................................... 2-33

Exceptions Raised by the CORBA Name Service .......................................... 2-33
AlreadyBound .......................................................... 2-34
CannotProceed .......................................................... 2-35
InvalidAddress ......................................................... 2-36
InvalidName ........................................................... 2-37
NotEmpty .............................................................. 2-38
NotFound ............................................................... 2-39

3. Managing a WebLogic Enterprise Namespace

Installing the CORBA Name Service ......................................................... 3-1
Starting the Server Process for the CORBA Name Service ................. 3-2
Making the Namespace Persistent ......................................................... 3-3
Compressing the Persistent Storage File ............................................. 3-4
Removing Orphan NamingContext Objects ........................................ 3-5
Federating the Namespace ...................................................................... 3-6
Inbound Federation .................................................................................. 3-6
Outbound Federation .............................................................................. 3-7
Federation Across WebLogic Enterprise Domains ................................ 3-8
Managing Binding Iterators ........................................................................ 3-9
Using the CORBA Name Service in Secure WebLogic Enterprise
Applications ......................................................................................... 3-9
4. Developing an Application that Uses the CORBA Name Service

   Development Steps....................................................................................................... 4-2
   Step 1: Obtain the OMG IDL for the CosNaming Interfaces................................. 4-3
   Step 2: Include the Client Stub for the CosNaming Interfaces ............................. 4-6
   Step 3: Connect to the WebLogic Enterprise Namespace................................. 4-7
   Step 4: Bind an Object to the WebLogic Enterprise Namespace......................... 4-9
   Step 5: Use a Name to Locate an Object in the WebLogic Enterprise Namespace

5. Using the Name Service Sample Application

   How the Name Service Sample Application Works ........................................... 5-1
   Building and Running the Name Service Sample Application ......................... 5-3
     Step 1: Copy the Files for the Name Service Sample Application into a Work Directory ................................................................................ 5-4
     Step 2: Change the Protection Attribute on the Files for the Name Service Sample Application.......................................................... 5-7
     Step 3: Verify the Settings of the Environment Variables................................. 5-8
     Step 4: Execute the runme Command................................................................. 5-10
   Using the Name Service Sample Application.................................................. 5-16
About This Document

This document provides information on using the BEA WebLogic Enterprise™ CORBA Name Service.

This document covers the following topics:

- Chapter 1, “Overview of the CORBA Name Service,” introduces the features and concepts of the WebLogic Enterprise CORBA Name Service.
- Chapter 2, “CORBA Name Service Reference,” describes the commands and application programming interfaces (APIs) of the WebLogic Enterprise CORBA Name Service.
- Chapter 3, “Managing a WebLogic Enterprise Namespace,” describes the administration tasks associated with the WebLogic Enterprise CORBA Name Service.
- Chapter 4, “Developing an Application that Uses the CORBA Name Service,” explains developing a WebLogic Enterprise application that uses a Namespace to store references to objects.
- Chapter 5, “Using the Name Service Sample Application,” describes how to build and run the Name Service sample application.

What You Need to Know

This document is intended for programmers who are developing applications with the WebLogic Enterprise product and want to use the name service feature.
e-docs Web Site

The BEA WebLogic Enterprise product documentation is available on the BEA System, Inc. corporate Web site. From the BEA Home page, click the Product Documentation button or go directly to the “e-docs” Product Documentation page at http://e-docs.bea.com.

How to Print the Document

You can print a copy of this document from a Web browser, one file at a time, by using the File—>Print option on your Web browser.

A PDF version of this document is available on the WebLogic Enterprise documentation Home page on the e-docs Web site (and also on the documentation CD). You can open the PDF in Adobe Acrobat Reader and print the entire document (or a portion of it) in book format. To access the PDFs, open the WebLogic Enterprise documentation Home page, click the PDF Files button, and select the document you want to print.

If you do not have Adobe Acrobat Reader installed, you can download it for free from the Adobe Web site at http://www.adobe.com/.

Related Information

For more information about CORBA, Java 2 Enterprise Edition (J2EE), BEA Tuxedo®, distributed object computing, transaction processing, C++ programming, and Java programming, see the WebLogic Enterprise Bibliography in the WebLogic Enterprise online documentation.
Contact Us!

Your feedback on the BEA WebLogic Enterprise documentation is important to us. Send us e-mail at docsupport@bea.com if you have questions or comments. Your comments will be reviewed directly by the BEA professionals who create and update the WebLogic Enterprise documentation.

In your e-mail message, please indicate that you are using the documentation for the BEA WebLogic Enterprise 5.1 release.

If you have any questions about this version of BEA WebLogic Enterprise, or if you have problems installing and running BEA WebLogic Enterprise, contact BEA Customer Support through BEA WebSUPPORT at www.bea.com. You can also contact Customer Support by using the contact information provided on the Customer Support Card, which is included in the product package.

When contacting Customer Support, be prepared to provide the following information:

- Your name, e-mail address, phone number, and fax number
- Your company name and company address
- Your machine type and authorization codes
- The name and version of the product you are using
- A description of the problem and the content of pertinent error messages

### Documentation Conventions

The following documentation conventions are used throughout this document.

<table>
<thead>
<tr>
<th>Convention</th>
<th>Item</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>boldface text</strong></td>
<td>Indicates terms defined in the glossary.</td>
</tr>
<tr>
<td>Ctrl+Tab</td>
<td>Indicates that you must press two or more keys simultaneously.</td>
</tr>
<tr>
<td>Convention</td>
<td>Item</td>
</tr>
<tr>
<td>------------</td>
<td>------</td>
</tr>
<tr>
<td><em>italics</em></td>
<td>Indicates emphasis or book titles.</td>
</tr>
</tbody>
</table>
| **monospace** text | Indicates code samples, commands and their options, data structures and their members, data types, directories, and filenames and their extensions. Monospace text also indicates text that you must enter from the keyboard.  
*Examples*:  
```  
#include <iostream.h> void main ( ) the pointer psz  
chmod u+w *  
\tux\data\ap  
.doc  
tux.doc  
BITMAP  
float  
``` |
| **monospace boldface text** | Identifies significant words in code.  
*Example*:  
```  
void commit ( )  
``` |
| **monospace italic text** | Identifies variables in code.  
*Example*:  
```  
String expr  
``` |
| **UPPERCASE TEXT** | Indicates device names, environment variables, and logical operators.  
*Examples*:  
```  
LPT1  
SIGNON  
OR  
``` |
| **{}** | Indicates a set of choices in a syntax line. The braces themselves should never be typed.  
*Example*:  
```  
buildobjclient [-v] [-o name ] [-f file-list]...  
[-l file-list]...  
``` |
### Documentation Conventions

<table>
<thead>
<tr>
<th>Convention</th>
<th>Item</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Separates mutually exclusive choices in a syntax line. The symbol itself should never be typed.</td>
</tr>
<tr>
<td>...</td>
<td>Indicates one of the following in a command line:</td>
</tr>
<tr>
<td></td>
<td>That an argument can be repeated several times in a command line</td>
</tr>
<tr>
<td></td>
<td>That the statement omits additional optional arguments</td>
</tr>
<tr>
<td></td>
<td>That you can enter additional parameters, values, or other information</td>
</tr>
<tr>
<td></td>
<td>The ellipsis itself should never be typed.</td>
</tr>
<tr>
<td>Example:</td>
<td>buildobjclient [-v] [-o name ] [-f file-list]... [-l file-list]...</td>
</tr>
<tr>
<td>.</td>
<td>Indicates the omission of items from a code example or from a syntax line.</td>
</tr>
<tr>
<td>.</td>
<td>The vertical ellipsis itself should never be typed.</td>
</tr>
</tbody>
</table>
CHAPTER

1 Overview of the CORBA Name Service

This topic includes the following sections:

- The CORBA Name Service
- Understanding the CORBA Name Service

The CORBA Name Service

The WebLogic Enterprise Name Service (referred to throughout this document as the CORBA Name Service) allows WebLogic Enterprise CORBA server applications to advertise object references using logical names. WebLogic Enterprise CORBA client applications can then locate an object by asking the CORBA Name Service to look up the name.

The CORBA Name Service provides:

- An implementation of the Object Management Group (OMG) Interoperable Name Service (INS) specification.
- Application programming interfaces (APIs) for mapping object references into an hierarchical naming structure (referred to as a namespace).
- Commands for displaying bindings and for binding and unbinding naming context objects and application objects into the namespace.
Overview of the CORBA Name Service

The CORBA Name Service is a layered product. The CORBA Name Service is installed as part of the WebLogic Enterprise product. For a complete description of the supported platforms and the installation procedure, see the BEA WebLogic Enterprise Installation Guide.

When using the CORBA Name Service:

1. WebLogic Enterprise CORBA server applications bind a name to one of its application objects or a naming context object within a namespace.

2. WebLogic Enterprise CORBA client applications can then use the namespace to resolve a name and obtain an object reference to the application object or the naming context object.

Figure 1-1 presents an overview of the CORBA Name Service.

Figure 1-1  CORBA Name Service
Figure 1-2 shows how a namespace might be used to store objects that make up an order entry application.

**Figure 1-2  A WebLogic Namespace**

The illustrated application organizes its namespace by geographic region, then by department. To implement the namespace using the objects in the CORBA Name Service, each shadowed box would be implemented by a NamingContext object. A NamingContext object contains a list of CosNaming::Name data structures that have been bound to application objects or to other NamingContext objects. NamingContext objects are traversed to locate a particular name. For example, the logical name California.Manufacturing.Order can be used to locate the Order object.
Overview of the CORBA Name Service

A CosNaming::Name data structure is not simply a string of alphanumeric characters; it is a sequence of one or more CosNaming::NameComponent data structures. Each CosNaming::NameComponent data structure contains two strings, id and kind. The CORBA Name Service does not interpret or manage these strings, except to ensure that each ID is unique within a given NamingContext object.

WebLogic Enterprise CORBA server applications use the bind() method of the NamingContext object to bind a name to an application object contained in the server application. WebLogic Enterprise CORBA client applications use the resolve() method of a NamingContext object to locate an object using a binding.

The CORBA Name Service also provides a BindingIterator object and a NamingContextExt object. The BindingIterator object allows a client application to obtain a specified number of bindings in each call. The NamingContextExt object provides methods to use Uniform Resource Locators (URL) and stringified names.

For a complete description of the objects in the CORBA Name Service and their interfaces, see Chapter 2, “CORBA Name Service Reference.”
This topic includes the following sections:

- CORBA Name Service Commands
- Capabilities and Limitations of the CORBA Name Service
- Getting the Initial Reference to the NameService Environmental Object
- The CosNaming Data Structures Used by the CORBA Name Service
- The NamingContext Object
- The NamingContextExt Object
- The BindingIterator Object
- Exceptions Raised by the CORBA Name Service

**CORBA Name Service Commands**

The CORBA Name Service provides the following commands to manage the server process for the CORBA Name Service, bind and unbind objects to names in the namespace, and display the contents of the namespace:

- `cns`
- `cnsbind`
The following sections describe these commands.
**cns**

Controls the server process for the CORBA Name Service.

**Synopsis**

```
cns CLOPT="[-A] [servopts options] --
   [-b bucketcount]
   [-c]
   [-d]
   [-f filename]
   [-M maxiterators]
   [-p [persiststoragefilename] ]
```

**Description**

The server process for the CORBA Name Service provides a CORBA CosNaming compliant name service. You need to define the server process for the CORBA Name Service and its options in the `UBBCONFIG` file for your WebLogic Enterprise application as you do any other server process used by your WebLogic Enterprise application. Enter the `cns` command-line options after the double dash (`--`) in the `CLOPT` parameter of the `UBBCONFIG` file. The command-line options are as follows:

- **-b bucketcount**
  Specifies the hash table bucket count used internally by the server process to locate naming contexts in-memory. Each naming context has its own hash table. If your WebLogic Enterprise application uses a small number of bindings in each naming context, use a small bucket count (for example, 4 or 5). If your WebLogic Enterprise application uses a large number of bindings (for example, 1,000) in each naming context, use a larger number such as 50 for the bucket count.

- **-c**
  Compresses the persistent storage file when the server process for the CORBA Name Service starts. Over time the persistent storage file can grow in size as naming context and application objects are added and removed from the namespace. Compression reduces the size of the persistent storage file to a minimum. Dangling bindings are removed during compression. Dangling bindings are left in the namespace after the object the binding is associated with is deleted from the namespace. The `-p` command-line option must be specified when specifying the `-c` command-line option.

- **-d**
  Directs the server process for the CORBA Name Service to delete orphan contexts when the server process starts. An orphan context is a context that is not bound to any other context. It may never have been bound or it may have been bound to a context and the binding was destroyed either explicitly or as
a side-effect of a rebind. The \texttt{-p} command-line option must be specified when specifying the \texttt{-d} command-line option.

\texttt{-f filename}

Specifies a file into which the server process for the CORBA Name Service writes the Interoperable Object Reference (IOR) of the root of the namespace.

\texttt{-M maxiterators}

Defines the maximum number of binding iterators that can be outstanding at any one time.

Binding iterators are created when a client application uses the \texttt{CosNaming::NamingContext::list()} method. The client application should use the \texttt{CosNaming::BindingIterator::destroy()} method to delete a binding iterator when the client application is done using the binding iterator.

If a client application does not specifically delete binding iterators, the server process for the CORBA Name Service deletes the binding iterators when the number reaches the value specified in the \texttt{-M} command-line option. Once the maximum number of binding iterators is reached, any attempt to create a new binding iterator causes the server process for the CORBA Name Service to destroy a binding iterator currently in use by the client application.

Binding iterators are deleted using a least-recently-used algorithm. The default value is 20. A value of 0 indicates that there is no maximum number of binding iterators (meaning binding iterators are never destroyed by the server process for the CORBA Name Service and the associated memory is not released). If a value of 0 is specified, the client application must explicitly use the \texttt{CosNaming::BindingIterator::destroy()} method to delete outstanding binding iterators.

\texttt{-p [persistentstoragefilename]}

Directs the server process for the CORBA Name Service to save a copy of the current namespace to persistent storage using the specified file. If a filename is not specified, the value of the \texttt{CNS\_PERSIST\_FILE} environmental variable is used. If the \texttt{CNS\_PERSIST\_FILE} environment variable is not set, the following files are used:

**Windows NT**

\%APPDIR\%\cnspersist.dat
UNIX

$APPDIR/cnspersist.dat

The persistent storage file is read when the server process for the CORBA Name Service starts. The persistent storage file is added to as changes are made to the namespace. If you want to create a new namespace, the existing persistent storage file must be deleted or a new one must be created on the server process for the CORBA Name Service.
cnsbind

Binds application objects and naming context objects into the namespace.

**Note:** The cnsbind command interacts with the CosNaming interfaces. The server process for the CORBA Name Service must be running to use this command.

**Synopsis**

```
cnsbind
    [-C]
    [-f root_context_filename]
    [-h]
    [-N]
    [-o ior_filename]
    [-r]
    [-T TObjAddr]
    bind_name
```

**Description**

The cnsbind command binds new application and naming context objects into the namespace using the CORBA CosNaming interfaces. This command facilitates the creation of a federated namespace. If an exception is returned when the cnsbind command is invoked, the command exits and an appropriate message is displayed.

The command-line options for the cnsbind command are as follows:

- **-C**
  Specifies that the cnsbind command creates a context using the bind_name for the name and the ior_filename specified for the -o command-line option. The -C command-line option is used to federate a naming context object from one namespace into the specified namespace.

- **-f root_context_filename**
  Specifies the file containing the IOR of the server process for the CORBA Name Service with which the command interacts to modify the contents of the namespace. If this command-line option is not specified, the command uses the Tobj_Bootstrap::resolve_initial_references() method with the NameService environmental object to locate the server process for the CORBA Name Service in the specified WebLogic Enterprise domain. The host and port in the IOR must match the value of TOBJADDR. This command-line option overrides the setting for the TOBJADDR environment variable. If the command-line option is not specified, the TOBJADDR environment variable is used.

- **-h**
  Prints the command syntax.
CORBA Name Service Commands

-N
  Creates a new context and binds the new context into the namespace using the
  specified name. The -o command-line option is not needed with the -N
  command-line option because the cnsbind command is creating a new
  context. If the -o command-line option is used with the -N command-line
  option, the information from the -o command-line option is ignored.

-o ior_filename
  Specifies a file that contains the IOR of the object to be bound into the
  namespace specified via the -f command-line option. If the -C
  command-line option is specified, an object of type ncontext is created
  otherwise a object of type nobject is created.

-r
  Creates a binding for an application or naming context object even if the name
  already has a binding. The default behavior of the cnsbind command
  without the -r command-line option is to raise the AlreadyBound exception
  in the case where a binding for the specified object already exists. If an
  AlreadyBound or any other exception is returned when the cnsbind
  command is invoked, the command exits and an “Error, already bound”
  message is displayed.

-T TObjAddr
  Specifies the host and port for a WebLogic Enterprise domain. Before
  connecting to a server process for the CORBA Name Service, the cnsbind
  command must log into the WebLogic Enterprise domain in which the server
  process is running. This command-line option overrides the setting for the
  TOBJADDR environment variable. If the command-line option is not specified,
  the TOBJADDR environment variable is used. The valid format for TOBJADDR
  is //hostname:port_number.

bind_name
  Specifies the name to be bound to the application object or name context
  object added to the namespace relative to either the root naming context
  retrieved via the Tobj_Bootstrap::resolve_initial_references() method, or the naming context identified by the stringified IOR obtained from
  the -f command-line option. The bind_name string should conform to the
  name string form specified in the Object Management Group (OMG)
  Interoperable Name Service (INS) specification.

Examples
  The following example illustrates binding an application object:

cnsbind -o ./app_obj_iyor.txt MyContext/AppObject1
The following example illustrates binding a naming context object:

cnsbind -N MyContext/CtxObject1

The following example illustrates binding a federation point to another namespace:

cnsbind -C -o ./remote_iortxt MyContext/RemoteNSCtx1
**cnsls**

Displays the contents of the namespace.

**Note:** The `cnsls` command interacts with the CosNaming interfaces. The server process for the CORBA Name Service must be running to use this command.

**Synopsis**

```
cnsls
    [-f root_context_filename]
    [-h]
    [-s]
    [-R]
    [-T TobjAddr]
    [resolve_name]
```

**Description**
The `cnsls` command displays the contents of the namespace using the CORBA CosNaming interfaces. If non-printing characters are used as part of a NameComponent data structure, the behavior of the `cnsls` command is undefined. If an exception is returned when the `cnsls` command is invoked, the command exits and an appropriate message is displayed.

The command-line options for the `cnsls` command are as follows:

- `-f root_context_filename`
  Specifies the file containing the IOR of the server process for the CORBA Name Service with which the command interacts to modify the contents of the namespace. If this command-line option is not specified, the command uses the Tobj_Bootstrap::resolve_initial_references() method with the NameService environmental object to locate the server process for the CORBA Name Service in the specified WebLogic Enterprise domain. The host and port in the IOR must match the value of TOBJADDR. This command-line option overrides the setting for the TOBJADDR environment variable. If the command-line option is not specified, the TOBJADDR environment variable is used.

- `-h`
  Prints the command syntax.

- `-s`
  Displays the stringified IOR for the namespace name specified in `resolve_name` command-line option.

- `-R`
  Recursively displays namespace bindings beginning at `resolve_name`. This command line option may cause the `cnsls` command to cross federation.
boundaries with no indication when such a boundary is cross. Also, if cycles exist in the namespace information, this command line option can cause the `cnsls` command to enter a loop.

-T TObjAddr
Specifies the host and port for a WebLogic Enterprise domain. Before connecting to a server process for the CORBA Name Service, the `cnsls` command must log into the WebLogic Enterprise domain in which the server process is running. This command-line option overrides the setting for the TOBJADDR environment variable. If the command-line option is not specified, the TOBJADDR environment variable is used.

-resolve_name
Specifies the name to resolve in the name service relative to either the root naming context retrieved via the `Tobj_Bootstrap::resolve_initial_references()` method or the naming context identified by the stringified IOR obtained from the `-f` command-line option. The `resolve_name` string should conform to the name string form specified in the OMG INS specification. The backslash (`\`) character is used to delimit name components and the period (`.`) character separates the `id` and `kind` fields.

If this command-line option is not specified, the root context is resolved.

Example
cnsls -R MyContext.kind/AnotherContext
[context] MyContext.kind/AnotherContext
[object] Obj1
[object] Obj2
[context] Ctx1
[object] AnotherObject
**cnsunbind**

Removes bindings from the namespace.

**Synopsis**

```
cnsunbind
    [-D]
    [-f root_context_filename]
    [-h]
    [-T TObjAddr]
    bind_name
```

**Description**

The `cnsunbind` command removes bindings from the namespace. If an exception is returned when the `cnsunbind` command is invoked, the command exits and an appropriate message is displayed.

The `cnsunbind` command-line options are as follows:

- **-D**
  Destroys the naming context bound to the `bind_name` after removing the binding. Specifying the `-D` command-line option when deleting a context prevents the context from being orphaned if it is not part of another binding. This command-line option should be used with care because it can cause dangling bindings (for example, if the binding was bound to multiple naming context objects at the same time).

- **-f root_context_filename**
  Specifies the file containing the IOR of the server process for the CORBA Name Service with which the command interacts to modify the contents of the namespace. If this command-line option is not specified, the command uses the `Tobj_Bootstrap::resolve_initial_references()` method with the NameService environmental object to locate the server process for the specified WebLogic Enterprise domain.

- **-h**
  Prints the command syntax.

- **-T TObjAddr**
  Specifies the host and port for a WebLogic Enterprise domain. Before connecting to a server process for the CORBA Name Service, the `cnsbind` command must log into the WebLogic Enterprise domain in which the server process is running. This command-line option overrides the setting for the `TOBJADDR` environment variable. If the command-line option is not specified, the `TOBJADDR` environment variable is used.
Using the CORBA Name Service

bind_name

Specifies the name of the binding to be removed from the namespace relative to either the root naming context retrieved via the `Tobj_Bootstrap::resolve_initial_references()` method or the naming context identified by the stringified IOR obtained from the `-f` command-line option. The `bind_name` string should conform to the name string form specified in the OMG INS specification.

Examples

The following example illustrates removing a binding from the namespace:

cnsunbind MyContext/CtxObject1

The following example illustrates removing a binding from the namespace and destroying the object to which it was bound:

cnsunbind -D MyContext/CtxObject1

Capabilities and Limitations of the CORBA Name Service

The CORBA Name Service has the following capabilities and limitations:

- A null character must only be used to terminate the `id` and `kind` strings (empty strings are considered valid).
- Wide characters are not supported.
- The CORBA Name Service imposes no limit on the length of the strings in a name component.
- The CORBA Name Service imposes no maximum on the number of components in a name. Zero length names are illegal.
- The CORBA Name Service imposes no limit on the number of bindings in a context.
- The CORBA Name Service imposes no limit on the number of bindings (implementation-wide).
- The CORBA Name Service imposes no limit on the number of contexts.
Getting the Initial Reference to the NameService Environmental Object

- The CORBA Name Service deletes orphaned naming contexts and dangling bindings when starting the server process for the CORBA Name Service.
- The CORBA Name Service deletes orphaned naming contexts when starting the server process for the CORBA Name Service.
- The CORBA Name Service offers the option of cleaning up binding iterator objects based on a least-recently-used algorithm. For more information, see “Managing Binding Iterators” on page 3-9.
- The CORBA Name Service does not throw the CannotProceed exception.

Getting the Initial Reference to the NameService Environmental Object

A NameService environmental object has been added for the purpose of connecting to the root of the namespace. When using the NameService environmental object, the Object Request Broker (ORB) locates the root of the namespace. Use the Bootstrap object to get an initial reference to the NameService environmental object. For more information, see “Step 3: Connect to the WebLogic Enterprise Namespace.”

The CosNaming Data Structures Used by the CORBA Name Service

The CORBA Name Service uses the following CosNaming data structures:
- CosNaming::BindingList
- CosNaming::BindingType
- CosNaming::Istring
- CosNaming::Name
- CosNaming::NameComponent
The NamingContext Object

The NamingContext object is used to contain and manipulate a list of names that are bound to Object Request Broker (ORB) objects or to other NamingContext objects. WebLogic Enterprise CORBA client applications use this interface to resolve or list all the names within that context. WebLogic Enterprise CORBA server applications use this object to bind names to application objects or naming context objects. Listing 2-1 shows the OMG IDL for the NamingContext object.

Listing 2-1  OMG IDL for the NamingContext Object

```idl
module CosNaming
    interface NamingContext {
        void bind(in Name, in Object obj)
            raises(NotFound, CannotProceed, InvalidName, AlreadyBound);
        void rebind(in Name, in Object obj)
            raises(NotFound, CannotProceed, InvalidName);
        void bind_context(in Name n, in NamingContext nc)
            raises(NotFound, CannotProceed, InvalidName, AlreadyBound);
        void rebind_context(in Name n, in NamingContext nc)
            raises(NotFound, CannotProceed, InvalidName)
        Object resolve(in Name n)
            ;
        void unbind(in Name n)
            raises(NotFound, CannotProceed, InvalidName);
        NamingContext new_context
            ;
        NamingContext bind_new_context(in Name n)
            raises(NotFound, CannotProceed, InvalidName, AlreadyBound);
        void destroy()
            raises(NotEmpty);
        void list(in unsigned long how_many,
            out BindingList bl,
            out BindingIterator bi);
    };
```
CosNaming::NamingContext::bind()

Synopsis  Attempts to bind the specified object to the specified name by resolving the context associated with the first NameComponent data structure and then binding the object to the new context.

C++ Mapping  void bind(in Name n, in Object obj);

Java Mapping  void bind (NameComponent [] n, Object obj)

Parameters  

n  A Name data structure, initialized with the desired name of the object.

obj  The object to bind to the supplied name.

Exceptions  

AlreadyBound  
The Name on a bind() or a bind_context() method has already been bound to another object within the naming context.

InvalidName  
The specified Name has zero name components or one of the first name components did not resolve to a naming context.

NotFound  
The Name or one of its components, could not be found.

Description  
Naming contexts bound with bind do not participate in name resolution when compound names are passed to be resolved.

Return Values  None.
CosNaming::NamingContext::bind_context()

**Synopsis**
This method is identical to the bind() method, except that the supplied Name is associated with a NamingContext object.

**C++ Mapping**
```cpp
void bind_context(in Name n, in NamingContext nc);
```

**Java Mapping**
```java
void bind_context (NameComponent [] n, NamingContext nc)
```

**Parameters**
- **n**
  A Name data structure initialized with the desired name for the naming context. The first NameComponent data structure in the sequence must resolve to a naming context.
- **nc**
  The NamingContext object to be bound to the supplied name.

**Exceptions**
- **AlreadyBound**
  The Name on a bind() or a bind_context() method has already been bound to another object within the naming context.
- **InvalidName**
  The specified Name has zero name components or one of the first name components did not resolve to a naming context.
- **NotFound**
  The Name or one of its components, could not be found.
- **BAD_PARAM**
  Indicates the call attempted to bind a nil context.

**Description**
Naming contexts bound with bind_context() participate in name resolution when compound names are passed to be resolved.

**Return Values**
None.
CosNaming::NamingContext::bind_new_context()

Synopsis  Creates a new context and binds it to the specified Name within this context.

C++ Mapping  NamingContext bind_new_context(in Name n);

Java Mapping  bind_new_context (NameComponent [] n)

Parameter  n

A Name data structure, initialized with the desired name for the newly created
NamingContext object.

Exceptions  AlreadyBound

The Name on a bind() or a bind_context() method has already been
bound to another object within the naming context.

InvalidName

The specified Name has zero name components or one of the first name
components did not resolve to a naming context.

NotFound

The Name or one of its components, could not be found.

Description  This method combines the CosNaming::NamingContext::new_context() and
CosNaming::NamingContext::bind_context() methods into a single method.

Return Values  Returns a reference to a new NamingContext object.
CosNaming::NamingContext::destroy()

Synopsis
Deletes a NamingContext object. Any subsequent attempt to invoke methods on the NamingContext object raises a CORBA::NO_IMPLEMENT exception.

C++ Mapping
void destroy();

Java Mapping
void destroy()

Parameter
None.

Exceptions
NotEmpty
If the NamingContext object contains bindings, the method raises NotEmpty.

Description
Before using this method, all name objects that have been bound to the NamingContext object should be unbound using the CosNaming::NamingContext::unbind() method.

Return Values
None.
CosNaming::NamingContext::list()

Synopsis
Returns all of the bindings contained by this naming context.

C++ Mapping
void list(in unsigned_long how_many,
          out BindingList bl,
          out BindingIterator bi);

Java Mapping
void list(int how_many,
          BindingListHolder bl,
          BindingIteratorHolder bi)

Parameters
   how_many
      The maximum number of bindings to be returned in the list.

   bl
      A list of returned bindings where each element is a Binding containing a Name of length 1 representing a single NameComponent object. The number of bindings in the list will not exceed how_many.

   bi
      A reference to a BindingIterator object for use in traversing the rest of the bindings.

Exceptions
   InvalidName
      The specified Name has zero name components or one of the first name components did not resolve to a naming context.

   NotFound
      The Name or one of its components, could not be found.

Description
This method returns a sequence of name bindings. If more name bindings exist than can fit in the bl list, a BindingIterator object is returned. The BindingIterator object can be used to get the next set of bindings. The BindingIterator object may return less than the requested number of bindings if it is at the end of the list. If bi returns a nil reference, then bl contains all of the remaining bindings.

Return Values
None.
**CosNaming::NamingContext::new_context()**

**Synopsis**  
Creates a new naming context. The newly created context is initially not bound to any Name.

**C++ Mapping**  
NamingContext new_context();

**Java Mapping**  
NamingContext new_context()

**Parameter**  
None.

**Exceptions**  
None.

**Description**  
Use the `CosNaming::NamingContext::bind_context()` method to bind the new naming context to a Name.

**Return Values**  
Returns a reference to a new naming context.
CosNaming::NamingContext::rebind()

Synopsis
This method is exactly the same as the bind() method, except that the Already-Bound except is never raised. If the specified Name has already been bound to another object, that binding is replaced by the new binding.

C++ Mapping
void rebind(in Name n, in Object obj);

Java Mapping
void rebind(NameComponent [] n, Object obj)

Parameters
n
A Name data structure, initialized with the desired name for the object.

obj
The object to be named.

Exceptions
InvalidName
The specified Name data structure has zero name components or one of the first name components did not resolve to a naming context.

NotFoundError
The Name or one of its components, could not be found. If this exception is raised because the binding already exists or the binding is of the wrong type, the rest_of_name member of the exception has a length of 1.

Description
Naming contexts bound with the rebind() method do not participate in name resolution when compound names are passed to be resolved.

Return Values
None.
CosNaming::NamingContext::rebind_context()

Synopsis
This method is exactly the same as the bind_context() method, except that the
AlreadyBound except is never raised. If the specified Name has already been bound
to another object, that binding is replaced by the new binding.

C++ Mapping
void rebind_context(in Name n, in NamingContext nc);

Java Mapping
void rebind_context(NameComponent [] n, NamingContext nc)

Parameters
n
A Name data structure, initialized with the desired name for the object.

nc
The NamingContext object to be rebound.

Exceptions
InvalidName
The specified Name data structure has zero name components or one of the
first name components did not resolve to a naming context.

NotFound
The component of a name does not identify a binding or the type of binding
is incorrect for the operation being performed. If this exception is raised
because a binding already exists or it is of the wrong type, the rest_of_name
member of the exception has a length of 1.

Description
Naming contexts bound with the rebind_context() method do not participate in
name resolution when compound names are passed to be resolved.

Return Values
None.
CosNaming::NamingContext::resolve

Synopsis
Attempts to resolve the specified Name.

C++ Mapping
Object resolve(in Name n);

Java Mapping
Object resolve (NameComponent n)

Parameters
n
A Name data structure, initialized with the desired name for the object.

Exceptions
InvalidName
The specified Name data structure has zero name components or one of the first name components did not resolve to a naming context.

NotFound
The component of a name does not identify a binding or the type of binding is incorrect for the operation being performed.

Description
The specified Name must exactly match the name used to bind the object. The CORBA Name Service does not return the type of the object. Client applications are responsible for narrowing the object to the appropriate type.

Return Values
Returns the object reference for the specified Name.
CosNaming::NamingContext::unbind

Synopsis
Performs the inverse operation of the bind() method, removing the binding associated with the specified Name.

C++ Mapping
void unbind(in Name n);

Java Mapping
void unbind (NameComponent [] n)

Parameters
n
A Name data structure, initialized with the desired name for the object.

Exceptions
InvalidName
The specified Name data structure has zero name components or one of the first name components did not resolve to a naming context.

NotFound
The component of a name does not identify a binding or the type of binding is incorrect for the operation being performed.

Description
This method removes the binding between a name and an object. It does not delete the object. Use the CosNaming::NamingContext::unbind() method and then the CosNaming::NamingContext::destroy() method to delete the object.

Return Values
None.

The NamingContextExt Object

The NamingContextExt object provides methods to use URLs and stringified names in the CORBA Name Service. The NamingContextExt object is derived from the NamingContext object. Note that the root of the CORBA Name Service is a NamingContextExt object (which means the root is also a NamingContext object). No special operation is needed to obtain a reference to a NamingContextExt object. Listing 2-2 shows the OMG IDL for the NamingContextExt object.
The NamingContextExt Object

Listing 2-2  OMG IDL for the NamingContextExt Object

module CosNaming {
    interface NamingContextExt : NamingContext {
        typedef string StringName;
        typedef string Address;
        typedef string URLString;

        StringName to_string(in Name n)
            raises(InvalidName);
        Name to_name(in StringName sn)
            raises(InvalidName);

        exception InvalidAddress {};

        URLString to_url(in Address addr, in StringName sn)
            raises(InvalidAddress, InvalidName);
        Object resolve_str(in StringName n)
            raises(NotFound, CannotProceed, InvalidName, AlreadyBound);
    }
};
CosNaming::NamingContextExt::resolve_str()

Synopsis  Takes a stringified name, converts it to a Name, and resolves it.

Syntax    object resolve_str(in StringName n);

Parameter n
          The stringified name to be resolved.

Exceptions
           InvalidName
          The name is invalid. A name of length zero is invalid.

           NotFound
          The component of the name does not identify a binding or the type of the
          binding is incorrect for the operation being performed.

Description  This is a convenience method that performs a resolve in the same manner as the Cos-
             Naming:NamingContext::resolve() method. The method accepts a stringified
             name as an argument instead of a Name object. The method returns errors if the string-
             ified name is invalid or if the method cannot bind it.

Return Values  A reference to the bound name.
CosNaming::NamingContextExt::to_name()

Synopsis  Takes a stringified name and returns a Name object.

Syntax    Name to_name(in StringName sn);

Parameter  

    sn

The stringified name to be resolved to a Name object.

Exceptions  InvalidName

    The name is invalid. A name of length zero is invalid.

Description  This method accepts a stringified name and returns a Name object. The method returns errors if the name is invalid.

Return Values  Returns a Name object.
CosNaming::NamingContextExt::to_string()

Synopsis  Accepts a Name object and returns a stringified name.

Syntax    StringName to_string(in Name n);

Parameter n
    The Name object to be converted to stringified name

Exceptions InvalidName
    The name is invalid. A name of length zero is invalid.

Description This method accepts a Name object and returns a stringified name. It returns errors if the name is invalid.

Return Values Returns a stringified name.
The BindingIterator Object

CosNaming::NamingContextExt::to_URL()

Synopsis
Combines a URL and a stringified name and returns a URL string.

Syntax
URLString to_URL(in Address addr, in StringName sn);

Parameter
addr
A URL. If this parameter is not defined, the local host name is used with the
IIOP protocol.

sn
The stringified name to be combined with the URL.

Exceptions
InvalidAddress
The URL is invalid.

InvalidName
The name is invalid. A name of length zero is invalid.

Return Values
Returns a URL string that combines the URL and the stringified name.

The BindingIterator Object

The BindingIterator object allows a client application to walk through the
unbounded collection of bindings returned by the list() method of a NamingCon-
text object. Using the BindingIterator object, a client application can control the
number of bindings obtained with each call. If a naming context is modified between
calls to the methods of a BindingIterator object, the behavior of further calls to
the next_one() method or the next_n() method is implementation specific.

If a client application creates BindingIterator objects but never calls the
destroy() method, the client application can run out of resources. The CORBA
Name Service is free to destroy binding iterators at any time and without warning to
the client application. Client applications should be written to expect the
OBJECT_NOT_EXIST exception from calls to a BindingIterator object and to han-
dle this exception gracefully.

Listing 2-3 shows the OMG IDL for the BindingIterator object.
Listing 2-3  OMG IDL for BindingIterator Object

module CosNaming {
    interface BindingIterator {
        boolean next_one(out Binding b);
        boolean next_n(in unsigned long how_many,
                        out BindingList b);
        void destroy();
    }
};
The BindingIterator Object

CosNaming::BindingIterator::destroy()

Synopsis
Destroys the BindingIterator object and releases the memory associated with the object. Failure to call this method results in increased memory usage.

C++ Mapping
void destroy();

Java Mapping
void destroy();

Parameter
None.

Exceptions
None.

Description
If a client application invokes any operation on a BindingIterator object after calling the destroy() method, the operation raises an OBJECT_NOT_EXIST exception.

Return Values
None.
CosNaming::BindingIterator::next_n()

Synopsis
Returns a BindingList data structure containing the number of requested bindings from the list. The number of bindings returned may be less than the requested amount if the list is exhausted.

C++ Mapping
boolean next_n(in unsigned_long how_many, out BindingList bl);

Java Mapping
boolean next_n(int how_many, BindingListHolder bl);

Parameter
how_many
The maximum number of bindings to return.

bl
A BindingList data structure containing no more than the requested number of bindings.

Exceptions
BAD_PARAM
Raised if the how_many parameter has a value of zero.

Return Values
CORBA::FALSE is returned when the list has been exhausted. Otherwise, CORBA::TRUE is returned.
exceptions Raised by the CORBA Name Service

CosNaming::BindingIterator::next_one()

Synopsis  Returns the next Binding object in the list.

C++ Mapping  boolean next_one(out Binding b);

Java Mapping  boolean next_one(BindingHolder b);

Parameter  b  The next Binding object from the list.

Exceptions  None.

Return Values  CORBA::FALSE is returned when the list has been exhausted. Otherwise, CORBA::TRUE is returned.

Exceptions Raised by the CORBA Name Service

This section describes the exceptions raised by the CORBA Name Service.
AlreadyBound

Syntax

    exception AlreadyBound{};

Parameter

    None.

Description

    This exception is raised when an object is already bound to the supplied name. Only one object can be bound to a name in a context.
CannotProceed

Syntax
exception CannotProceed{};

Parameters
NamingContext ctx
    The context that the operation may be able to retry from.

Name rest_of_name
    The remainder of the non working name.

Description
This exception is raised when an unexpected exception is encountered and the method cannot proceed in a meaningful way.
InvalidAddress

Syntax   exception InvalidAddress{};
Parameter None.
Description This exception is raised if a URL is invalid.
### InvalidName

**Syntax**
```
exception InvalidName{};
```

**Parameter**
None.

**Description**
This exception is raised if a Name is invalid. A name length of zero is invalid.
NotEmpty

Syntax  exception NotEmpty{};

Parameter  None.

Description  This exception is raised when the destroy() method is used on a NamingContext object that contains bindings. A NamingContext object must be empty before it is destroyed.
Exceptions Raised by the CORBA Name Service

**NotFoundException**

**Syntax**

```plaintext
exception NotFound(NotFoundReason why; Name rest_of_name);
```

**Parameters**

- **why**
  
  The context that the operation may be able to retry from.

- **rest_of_name**
  
  The remainder of the nonworking name.

**Description**

This exception is raised when a component of the name does not identify a binding, or if the type of binding is incorrect for the operation being performed. The `why` parameter explains the reason for the error. The `rest_of_name` parameter identifies the cause of the error. The following causes can appear:

- **missing_node**—the first name component in the `rest_of_name` parameter is a binding that is not bound under that name within its parent context.

- **not_context**—the first name component in the `rest_of_name` parameter is a binding with a type of `nobject` when the type of `ncontext` was required.

- **not_object**—the first name component in the `rest_of_name` parameter is a binding with a type of `ncontext` when the type of `nobject` was required.
CHAPTER 3

Managing a WebLogic Enterprise Namespace

This topic includes the following sections:

- Installing the CORBA Name Service
- Starting the Server Process for the CORBA Name Service
- Making the Namespace Persistent
- Compressing the Persistent Storage File
- Removing Orphan NamingContext Objects
- Federating the Namespace
- Managing Binding Iterators

Installing the CORBA Name Service

You install the CORBA Name Service when you install the WebLogic Enterprise product. For complete information about installing the WebLogic Enterprise product, see the BEA WebLogic Enterprise Installation Guide.
Starting the Server Process for the CORBA Name Service

To start the server process for the CORBA Name Service, you need to define the server process in the UBBCONFIG file for your WebLogic Enterprise application. Use the cns command to start the server process for the CORBA Name Service. List the cns command-line options after the CLOPT parameter in the UBBCONFIG file. Note there can be only one CORBA Name Service server process running per WebLogic Enterprise domain. Listing 3-1 shows an example of the UBBCONFIG entry for the server process for the CORBA Name Service.

Listing 3-1 UBBCONFIG File Entry for CORBA Name Service

```
... 
# Server process for WLE CORBA Name Service 
#
"drive:\ns\server\cns.exe SRVGRP = SYS_GRP
SRVID = 6
RESTART = N
CLOPT = "-A -- -f drive:\cnsroot.dat -M 0"
```

For a complete description of the cns command and its options, see Chapter 2, “CORBA Name Service Reference.” For information about creating a configuration file, see the Administration Guide.

Once the server process for the CORBA Name Service is started, you can use the commands listed in Table 3-1 to display the contents of the namespace and manage objects in the namespace. For a complete description of the commands and their options, see Chapter 2, “CORBA Name Service Reference.”
Making the Namespace Persistent

The CORBA Name Service keeps two copies of the information in a namespace. One copy is kept in-memory. Access to this copy is fast and optimized for name resolution. The other copy is optionally saved to persistent storage allowing the state and structure of the namespace to be saved and restored.

The primary goal of making a namespace persistent is to keep a current representation of the in-memory naming information maintained by the currently running instance of the namespace. By maintaining a persistent copy of the namespace, the CORBA Name Service can recreate current naming information in case the server process of the CORBA Name Service is terminated. A new instance of the server process for the CORBA Name Service can be configured to read the persistent storage file to recreate the last recorded naming information.

To create a persistence copy of the namespace and store the copy to a file, specify the -p option of the cns command when starting the server process for the CORBA Name Service. The CORBA Name Service creates a persistent storage file with the specified location and name.

### Table 3-1 Commands for Managing a WebLogic Enterprise Namespace

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cns</td>
<td>Starts the server process for the WebLogic Enterprise namespace.</td>
</tr>
<tr>
<td>cnsbind</td>
<td>Binds application objects and naming context objects to the WebLogic Enterprise namespace.</td>
</tr>
<tr>
<td>cnsls</td>
<td>Displays the contents of a WebLogic Enterprise namespace.</td>
</tr>
<tr>
<td>cnsunbind</td>
<td>Removes bindings from a WebLogic Enterprise namespace.</td>
</tr>
</tbody>
</table>
If the persistent storage file specified by the `-p` option already exists, the file is opened and processed. A backup of the persistent storage file is always made prior to the startup of the server process for the CORBA Name Service. The name of the backup copy of the persistent storage file is `filename.BAK`. If you want to reuse the name of the persistent storage file, you must delete or move the existing file and then restart the server process for the CORBA Name Service.

If the persistent storage file is successfully created, an entry for the file is written to the `ULOG` file. The entry indicates the directory location and name of the file, whether or not the file was newly created, and the mechanism used to determine the name of the file (for example, specified, environmental, or default). If an error occurs when creating the persistent storage file, an entry is written to the `ULOG` file indicating the type of error that occurred.

Since the server process for the CORBA Name Service recreates the structure of the namespace from the persistent storage file at startup, the startup time is directly proportional to the size of the persistent storage file. Very large persistent storage files (on the order of hundreds of megabytes) can result in the server process for the CORBA Name Service taking several seconds or even minutes to recreate the namespace at startup.

## Compressing the Persistent Storage File

The persistent storage file contains information about all operations affecting the in-memory copy of the namespace. Over time, the persistent storage file can contain more information than is necessary to recreate the structure and state of the current namespace. In fact, the persistent storage file can grow quite large even though the structure of the namespace stays the same size.

The CORBA Name Service allows you to compress the persistent storage file to remove unneeded information. The `-c` option of the `cns` command controls compression of the persistent storage file. The compression option processes the current information to produce a new compressed persistent storage file.

When the server process for the CORBA Name Service is started, the compression operation performs the following:

1. Processes the in-memory structure of the namespace.
Removing Orphan NamingContext Objects

2. Overwrites the existing persistent storage file.

3. Deletes all bind and rebind entries which were removed from the namespace by unbind, rebind, or destroy operations.

4. Removes all dangling bindings. Dangling bindings are bindings left in the namespace after the object the binding is associated with is deleted from the namespace. Dangling bindings occur when a CosNaming::NamingContext::destroy() method is performed on a naming context without the naming context being unbound from its parent context.

The -c option can only be used if the -p option of the cns command is specified. For a complete description of the -c option of the cns command, see Chapter 2, “CORBA Name Service Reference.”

Removing Orphan NamingContext Objects

An orphan context is a context that is not bound to any other context. The context may have never been bound or it may have been bound and the binding was destroyed either explicitly or as the result of a rebind. In the CORBA Name Service, orphan NamingContext objects are created in one of the following ways:

- **Using the** CosNaming::NamingContext::new_context() **method to create a new NamingContext object but never binding the new NamingContext object to the namespace.**

- **Using the** CosNaming::NamingContext::rebind() or CosNaming::NamingContext::unbind() **methods to unbind the NamingContext object from their last parent NamingContext object but never destroying the NamingContext object.**

Client applications and other namespaces federated to the NamingContext object can perform operations on orphan NamingContext objects as long as they maintain the object reference to the orphan NamingContext object.

The current implementation of the namespace maintains the orphan NamingContext objects in a special LostandFoundContext object.
Use the `-d` option of the `cns` command to delete orphan `NamingContext` objects from the namespace. The `-d` option unbinds and destroys all `NamingContext` objects identified as orphaned.

The `-d` option can only be used if the `-p` option of the `cns` command is specified. For a complete description of the `-d` option of the `cns` command, see Chapter 2, “CORBA Name Service Reference.”

Federating the Namespace

The CORBA Name Service supports the concept of a federated namespace which means elements of a logical namespace may reside on multiple, disparate, and remote machines. In a federated namespace, a `NamingContext` object can be bound to one namespace using an object reference to a `NamingContext` object maintained by another namespace. The CORBA Name Service supports federation with implementations of the CORBA Name Service running on other machines as well as third-party name services. In order for the CORBA Name Service to federate with a third-party name service, the third-party name service must support the naming formats specified in the Object Management Group (OMG) Interoperable Name Service (INS) specification.

The following topics explain how to support inbound and outbound federation as well as federation with third-party name services.

Inbound Federation

Inbound federation is the ability to bind a `NamingContext` object which exists in a local Weblogic Enterprise namespace into a namespace on a remote name service. Once the namespaces are federated, the remote name service can perform operations on `NamingContext` objects in a the WebLogic Enterprise namespace. Inbound federation with a third-party name service is done using the Internet Inter-Orb Protocol (IIOP). Therefore, the IIOP Listener/Handler for the CORBA Name Service must be configured to support unofical IIOP connections.
To enable the unofficial connections on the IIOP Listener/Handler, use the `-C` parameter of the ISL command. The `-C` parameter determines how the IIOP Listener/Handler will behave when unofficial connections are made to it. To use inbound federation, specify the `warn` or `none` values for the `-C` parameter. A value of `warn` causes the IIOP Listener/Handler to log a message to the user log exception when an unofficial connection is detected; no exception will be raised. A value of `none` causes the IIOP Listener/Handler to ignore unofficial connections. For more information about the ISL command, see the *WebLogic Enterprise Commands and MIB Reference*.

Listing 3-2 shows an example of the `UBBCONFIG` entry for the IIOP Listener/Handler that supports inbound federation.

### Listing 3-2  UBBCONFIG File Entry for an IIOP Listener/Handler That Supports Inbound Federation

```plaintext
# Entry to start IIOP Listener/Handler
# that supports inbound federation
ISL
  SRVGRP = SYS_GRP
  SRVID = 5
  MIN = 1
  MAX = 1
  CLOPT = "-A -- -n //blotto:2470 –C none"
```

### Outbound Federation

Outbound federation is the ability to bind a `NamingContext` object which exists in a remote name service into the namespace of a CORBA Name Service. Operations can then be performed on this `NamingContext` object using the CORBA Name Service. Outbound federation with a third-party name service is done using IIOP. Therefore, the IIOP Listener/Handler for the CORBA Name Service must be configured to support outbound federation.
To enable outbound federation on the IIOP Listener/Handler, use the `-o` parameter of the ISL command. The `-o` parameter causes the IIOP Listener to allow outbound IIOP invocations to objects located in server applications not connected to a IIOP Handler. For more information about the ISL command, see the *WebLogic Enterprise Commands and MIB Reference*.

Listing 3-3 shows an example of the UBBCONFIG entry for the IIOP Listener/Handler that supports outbound federation.

### Listing 3-3 UBBCONFIG File Entry for an IIOP Listener/Handler That Supports Outbound Federation

```plaintext
ISL
    SRVGRP = SYS_GRP
    SRVID = 5
    MIN = 1
    MAX = 1
    CLOPT = "-A -- -n //blotto:2470 -O"
```

**Federation Across WebLogic Enterprise Domains**

Federation across multiple CORBA Name Service server processes running in different WebLogic Enterprise domains requires the use of Domain Gateways to allow for inter-domain communication. For more information about configuring a domain gateway, see the “Configuring Multiple Domains (WLE System)” section in the Administration topic.
Managing Binding Iterators

The OMG INS specification allows the collection of outstanding binding iterators. Since binding iterators require explicit destruction by client applications, there is the chance that binding iterators will not be deleted properly.

To minimize the possibility that the CORBA Name Service will run out of resources due to a large number of unused binding iterators, use the `-M` option of the `cns` command to set the maximum number of binding iterators in the name service. Once the limit has been reached, requests for new binding iterators may result in the destruction of outstanding binding iterators. The CORBA Name Service uses a least-recently-used algorithm to select which binding iterators are deleted.

If the server process for the CORBA Name Service is started with the `-M` option, the CORBA Name Service may destroy a binding iterator that is currently being used by a WebLogic Enterprise application so all WebLogic Enterprise applications need to be able to handle the `CORBA::OBJECT_NOT_EXIST` system exception when invoking on binding iterators.

Using the CORBA Name Service in Secure WebLogic Enterprise Applications

When using the `cnsls`, `cnsbind`, and `cnsunbind` commands in a secure WebLogic Enterprise application, you need to obtain the PrincipalAuthenticator object for the WebLogic Enterprise domain and logon to the domain with the appropriate security information.

In order for a secure logon to occur, the WebLogic Enterprise domain must be configured with a security level of `TOBJ_SYSAUTH` or `TOBJ_APPAUTH`. The username for the `cnsls`, `cnsbind`, and `cnsunbind` commands is `cnsutils`. You need to use the `tpusradd` command to create a user named `cnsutils`. Depending on the Security level specified for the WebLogic Enterprise domain, the user password and/or the domain password may be defined in the `UBBCONFIG` file in the `USER_AUTH` and `APP_PW` environment variables. If these environment variables are not set and the
WebLogic Enterprise domain has a security level of `TOBJ_SYSAUTH` or `TOBJ_APPAUTH`, the `cnsls`, `cnsbind`, and `cnsunbind` commands will prompt for a password.

For more information about security levels and defining users in the WebLogic Enterprise security environment, see *Using Security*.
CHAPTER

4 Developing an Application that Uses the CORBA Name Service

This topic includes the following sections:

- Development Steps
  - Step 1: Obtain the OMG IDL for the CosNaming Interfaces
  - Step 2: Include the Client Stub for the CosNaming Interfaces
  - Step 3: Connect to the WebLogic Enterprise Namespace
  - Step 4: Bind an Object to the WebLogic Enterprise Namespace
  - Step 5: Use a Name to Locate an Object in the WebLogic Enterprise Namespace
Table 4-1 outlines the process for developing WebLogic Enterprise applications that use the CORBA Name Service.

Table 4-1 Development Process

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Obtain the OMG IDL for the CosNaming interfaces.</td>
</tr>
<tr>
<td>2</td>
<td>Include the client stubs for the CosNaming interfaces.</td>
</tr>
<tr>
<td>3</td>
<td>Connect to the WebLogic Enterprise namespace.</td>
</tr>
<tr>
<td>4</td>
<td>Bind an object to the WebLogic Enterprise namespace.</td>
</tr>
<tr>
<td>5</td>
<td>Use a name to locate an object in the WebLogic Enterprise namespace.</td>
</tr>
</tbody>
</table>

Before performing the steps in this topic, you need to start the server process for the CORBA Name Service. For more information, see “Starting the Server Process for the CORBA Name Service” on page 3-2.

After performing the development steps in this topic, use the buildjavaserver or buildserver command and the buildobjclient command as you would normally to compile server and client applications that use the CORBA Name Service. For more information about the buildjavaserver, buildobjclient, and buildserver commands, see WebLogic Enterprise Commands and MIB Reference.
Step 1: Obtain the OMG IDL for the CosNaming Interfaces

A WebLogic Enterprise application accesses the CORBA Name Service using the interfaces defined in CosNaming.idl. This Object Management Group (OMG) Interface Definition Language (IDL) file defines the interfaces, COSnaming data structures, and exceptions used by the CORBA Name Service. CosNaming.idl is located in the following directory locations:

**Windows NT**

drive:\$TUXDIR\include\CosNaming.idl

**UNIX**

/usr/local/$TUXDIRdir/include/CosNaming.idl

Listing 4-1 shows the OMG IDL for CosNaming.idl. The same OMG IDL file is used by both CORBA C++ and Java applications.

### Listing 4-1  CosNaming.idl

```idl
#ifndef _COSNAMING_IDL_
#define _COSNAMING_IDL_

module CosNaming {
    #pragma prefix "omg.org/CosNaming"
    typedef string Istring;
    struct NameComponent {
        Istring id;
        Istring kind;
    };
    typedef sequence<NameComponent> Name;
    enum BindingType { nobject, ncontext };
    struct Binding {
        Name binding_name;
    }

    #endif
}
```

Using the CORBA Name Service  4-3
Developing an Application that Uses the CORBA Name Service

BindingType binding_type;

typedef sequence <Binding> BindingList;

interface BindingIterator;

interface NamingContext {
    enum NotFoundReason { missing_node, not_context, not_object };

    exception NotFound {
        NotFoundReason why;
        Name rest_of_name;
    };

    exception CannotProceed {
        NamingContext cxt;
        Name rest_of_name;
    };

    exception InvalidName{};

    exception AlreadyBound {};

    exception NotEmpty{};

    void bind(in Name n, in Object obj) raises(NotFound, CannotProceed, InvalidName, AlreadyBound);

    void rebind(in Name n, in Object obj) raises(NotFound, Cannot Proceed, InvalidName);

    void bind_context(in Name n, in NamingContext nc) raises(NotFound, CannotProceed, InvalidName, AlreadyBound);

    void rebind_context(in Name n, in NamingContext nc) raises(NotFound, CannotProceed, InvalidName);
Step 1: Obtain the OMG IDL for the CosNaming Interfaces

```idl
Object resolve (in Name n)
    raises(NotFound,
           CannotProceed,
           InvalidName);

void unbind(in Name n)
    raises(NotFound,
           CannotProceed,
           InvalidName);

NamingContext new_context();
NamingContext bind_new_context(in Name n)
    raises(NotFound,
           AlreadyBound,
           CannotProceed,
           InvalidName);

void destroy() raises(NotEmpty);
void list(in unsigned long how_many,
          out BindingList bl,
          out BindingIterator bi);

interface BindingIterator {
    boolean next_one(out Binding b);
    boolean next_n(in unsigned long how_many,
                    out BindingList bl);
    void destroy();
};

interface NamingContextExt: NamingContext {
    typedef string StringName;
    typedef string Address;
    typedef string URLString;

    StringName to_string(in Name n) raises(InvalidName);
    Name to_name(in StringName sn)
        raises(InvalidName);

    exception InvalidAddress {};

    URLString to_url(in Address addr, in StringName sn)
        raises(InvalidAddress, InvalidName);

    Object resolve_str(in StringName n)
        raises(NotFound,
               CannotProceed,
               InvalidName,
               AlreadyBound
    );
```

Using the CORBA Name Service 4-5
Step 2: Include the Client Stub for the CosNaming Interfaces

The client stubs for the CosNaming interfaces are compiled and included as part of the software kit for the CORBA Name Service.

- If you are creating a WebLogic Enterprise CORBA C++ application, the client stubs are located in \$TUXDIR/lib/libomg. You need to include the generated client stub in your WebLogic Enterprise application as follows:

  ```
  #include "CosNaming_c.h
  ```

- If you are creating a WebLogic Enterprise CORBA Java application, the client stubs are located in \$TUXDIR/udataobj/java/jdk/m3envobj.jar. You need to include the generated client stub in your WebLogic Enterprise application as follows:

  ```
  import org.omg.CosNaming.*;
  ```

The interfaces for the CORBA Name service are in the `org.omg.CosNaming` package.

If you are using a third-party object request broker (ORB), you need to compile the CosNaming interfaces using your IDL compiler and include or import them into your WebLogic Enterprise application.
Step 3: Connect to the WebLogic Enterprise Namespace

The Bootstrap object has been modified to support a NameService environmental object for the purpose of connecting to the root of the namespace. When using the NameService environmental object, the Object Request Broker (ORB) locates the root of the namespace. The object reference can then be narrowed to CosNaming::NamingContext or CosNamingContextExt. You need to connect to the WebLogic Enterprise namespace before binding objects into the namespace and resolving names in the namespace.

Listing 4-2 and Listing 4-3 include C++ and Java code that establishes communication with a WebLogic Enterprise namespace.

Listing 4-2  C++ Example of Connecting to a Namespace

```cpp
...  
Tobj_Bootstrap * bootstrap = new Tobj_Bootstrap (v_orb.in(), "");  
CORBA::Object_var var_nameservice_oref=  
    bootstrap.resolve_initial_references("NameService");  
root = CosNaming::NamingContext::_narrow (obj);  
...  
```

Listing 4-3  Java Example of Connecting to a Namespace

```java
...  
Tobj_Bootstrap bootstrap = new Tobj_Bootstrap(orb, "");  
org.omg.CORBA.Object NameServiceobj =  
    gBootstrapObjRef.resolve_initial_references("NameService");  
CosNaming.NamingContextExt ns_root =  
    CosNaming.NamingContextExtHelper.narrow (ns_obj);  
...  
```

A stringified object reference for the root of the namespace can also be used to connect to a namespace in a WebLogic Enterprise domain. In order to use a stringified object reference, the -f command-line option must be specified when starting the server.
process for the CORBA Name Service. The \(-f\) command-line option writes the stringified object reference to the CNS\_ROOT\_FILE environment variable or to one of the following locations:

**Windows NT**

\%APPDIR\%\cnsroot.dat

**UNIX**

\$APPDIR/cnsroot.dat

The stringified object reference for the root of the namespace does not change when the server process for the CORBA Name Service is started and stopped because stringified object reference is associated with a particular host machine rather than a particular server process. A stringified object reference that has been retrieved to communicate with one WebLogic Enterprise namespace cannot be used to communicate with another WebLogic Enterprise namespace.

Listing 4-4 and Listing 4-5 include C++ and Java code that establishes communication with a WebLogic Enterprise namespace using a stringified object reference.

**Listing 4-4  C++ Example of Using a Stringified Object Reference**

```cpp
...
Tobj_Bootstrap * bootstrap;
bootstrap = new Tobj_Bootstrap (v_orb.in(), "");
CORBA::Object_var obj = GetRefFromFile ("cnsroot.dat", v_orb);
root = CosNaming::NamingContext::narrow (obj);
...
```

**Listing 4-5  Java Example of Using a Stringified Object Reference**

```java
...
Tobj_Bootstrap bootstrap = new Tobj_Bootstrap(orb, "");
BufferedReader inFile =
    newBufferedReader(new FileReader ("cnsroot.dat"));
String root_ior_string = inFile.readLine ();
org.omg.CORBA.Object ns_obj =
    orb.string_to_object (root_ior_string);
CosNaming.NamingContextExt ns_root =
    CosNaming.NamingContextExtHelper.narrow (ns_obj);
...
```
Step 4: Bind an Object to the WebLogic Enterprise Namespace

If you choose to use a stringified object reference in a WebLogic Enterprise application that also uses security and transactions, please note the following restrictions:

1. The WebLogic Enterprise application must create a Bootstrap object and connect to the IIOP Listener/Handler before using the stringified object reference to connect to a WebLogic Enterprise Namespace. By calling the Bootstrap object first, the WebLogic Enterprise application establishes an official connection to the IIOP Listener/Handler.

   If a WebLogic Enterprise application does not first create a Bootstrap object, transactions and security cannot be used with any object retrieved from the namespace. Transactions and security require the use of an official connection.

2. If more than one IIOP Listener/Handler is defined in the UBBCONFIG file, the WebLogic Enterprise application must use the first IIOP Listener/Handler defined in the UBBCONFIG file by the TOBJADDR environment variable.

   The CORBA Name Service creates the stringified object reference for the root of the namespace, using the default IIOP Listener/Handler’s host and port. The first IIOP Listener/Handler defined in a UBBCONFIG file is considered the default IIOPListener/Handler. Using the default IIOP Listener/Handler causes all object references retrieved by the CORBA Name Service to be official connections. Transactions and security require the use of official connections.

Step 4: Bind an Object to the WebLogic Enterprise Namespace

There are two ways to bind an object to the WebLogic Enterprise namespace:

- The cnsbind command
- The bind() method of the CosNaming::NamingContext object

The cnsbind command can be used to bind application objects or naming context objects to the WebLogic Enterprise namespace. The server process for the CORBA Name Service must be started before using the cnsbind command. For a complete description of the cnsbind command, see Chapter 2, “CORBA Name Service Reference.”
Developing an Application that Uses the CORBA Name Service

Listing 4-6 and Listing 4-7 shows the C++ and Java code uses the bind() method of the CosNaming::NamingContext object. The code examples accept two parameters, representing the id and kind fields for a Name. These parameters initialize a Name for the SimpleFactory object and bind the SimpleFactory object to the namespace.

**Listing 4-6  C++ Example of Binding a Name to the WebLogic Enterprise Namespace**

```cpp
... //Establish the Name used to identify the SimpleFactory object //in the namespace.

 CosNaming::Name_var factory_name = new CosNaming::Name(1);
  factory_name->length(1);
  factory_name[(CORBA::ULong) 0].id =
    (const char * "simple_factory"; //Create an object reference for the SimpleFactory object

 s_v_factory_refer = TP::create_object_reference(
    _tc_SimpleFactory->id(),
    "simple_factory",
    CORBA::NVList::_nil());
//Get the NameService object reference. See Listing 4-2.
//Place the object reference for SimpleFactory in the namespace

root->bind(factory_name, s_v_fact_ref);
...
```

**Listing 4-7  Java Example of Binding a Name to the WebLogic Enterprise Namespace**

```java
... //Create an object reference for the SimpleFactory object

 org.omg.CORBA.object fact_ref =
   TP.create_object_reference(
     SimpleFactoryHelper.id() //Create an object reference for the SimpleFactory object

   "simple_factory",
   null
 );
```

4-10  Using the CORBA Name Service
Step 5: Use a Name to Locate an Object in the WebLogic Enterprise Namespace

...  
//Get the NameService object reference. See Listing 4-3.  
//Place the object reference for SimpleFactory in the namespace  
CosNaming.NameComponent[] factName =  
    ns_root.to_name("simple_factory");  
ns_root.bind(factName, fact_ref);  
...

Step 5: Use a Name to Locate an Object in the WebLogic Enterprise Namespace

Use the resolve() method of the CosNaming::NamingContext object to locate an object in a namespace in a WebLogic Enterprise domain. Listing 4-8 and Listing 4-9 shows the C++ and Java code that accepts two parameters, representing the id and kind fields for a Name. The code example then binds to a naming context, resolves the name, and obtains an object reference for the specified object.

Listing 4-8   C++ Example of Locating a Name in the WebLogic Enterprise Namespace

...  
//Establish the Name used to identify the SimpleFactory object  
//in the namespace.  
CosNaming::Name_var factory_name = new CosNaming::Name(1);  
factory_name->length(1);  
factory_name[(CORBA::ULong) 0].id =  
    (const char * "simple_factory");  
factory_name[(CORBA::ULong) 0].kind =  
    (const char *) "";

//Locate the SimpleFactory object in the namespace  
CORBA::Object_var v_simple_factory_oref =  
    root->resolve(*factory_name);  
SimpleFactory_var v_simple_factory_ref =  
    SimpleFactory::_narrow(v_simple_factory_oref.in());

//Use the reference obtained from the WLE CORBA Name Service to find  
the Simple object  

Using the CORBA Name Service 4-11
Developing an Application that Uses the CORBA Name Service

Listing 4-9    Java Example of Locating a Name in the WebLogic Enterprise Namespace

```java
Simple_var v_simple = v_simple_factory_ref->find_simple();
...

//Find the SimpleFactory object in the namespace via a string name
org.omg.CORBA.Object simple_fact_oref =
    ns_root.resolve_str("simple_factory");
SimpleFactory simple_factory_ref =
    SimpleFactoryHelper.narrow(simple_fact_oref);

//Find the Simple object
Simple simple = simple_factory_ref.find_simple();
...
```
CHAPTER 5

Using the Name Service Sample Application

This topic includes the following sections:

- How the Name Service Sample Application Works
- Building and Running the Name Service Sample Application

How the Name Service Sample Application Works

The Name Service sample application is a modification of the Simpapp sample application. The Name Service sample application uses a namespace to store the SimpleFactory object. The server application creates the SimpleFactory object and binds the object to the namespace. The client application connects to the namespace, resolves the name of the SimpleFactory object, and then invokes methods on the SimpleFactory. There are C++ and Java versions of the Name Service sample application. Figure 5-1 illustrates how the Name Service sample application works.
Using the Name Service Sample Application

Figure 5-1  The Name Service Sample Application

The Name Service sample application implements the CORBA interfaces listed in Table 5-1:

Table 5-1  CORBA Interfaces for the Name Service Sample Application

<table>
<thead>
<tr>
<th>Interface</th>
<th>Description</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>SimpleFactory</td>
<td>Creates object references to the Simple object</td>
<td>find_simple()</td>
</tr>
<tr>
<td>Simple</td>
<td>Converts the case of a string</td>
<td>to_upper()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>to_lower()</td>
</tr>
</tbody>
</table>

Listing 5-1 shows the simple.idl file that defines the CORBA interfaces in the Name Service sample application.
Building and Running the Name Service Sample Application

Listing 5-1  OMG IDL Code for the Name Service Sample Application

```idl
#pragma prefix "beasys.com"

interface Simple
{
    //Convert a string to lower case (return a new string)
    string to_lower(in string val);

    //Convert a string to upper case (in place)
    void to_upper(inout string val);
};

interface SimpleFactory
{
    Simple find_simple();
};
```

Building and Running the Name Service Sample Application

To build and run the Name Service sample application, complete the following steps:

1. Copy the files for the Name Service sample application into a work directory.
2. Change the protection attribute on the files for the Name Service sample application.
3. Verify the environment variables.
4. Execute the `runme` command.
5  Using the Name Service Sample Application

Step 1: Copy the Files for the Name Service Sample Application into a Work Directory

You need to copy the files for the Name Service sample application into a work directory on your local machine. The following sections detail the directory location and sources files for the C++ and Java versions of the Name Service sample application.

C++ Version of the Name Service Sample Application

The files for the C++ version of the Name Service sample application are located in the following directories:

Windows NT

drive:\WLEdir\samples\corba\cnssimpapp

UNIX

/usr/local/WLedir/samples/corba/cnssimpapp

You will use the files listed in Table 5-2 to build and run the C++ version of the Name Service sample application.

Table 5-2  Files Included in the C++ Version of the Name Service Sample Application

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simple.idl</td>
<td>The OMG IDL code that declares the Simple and SimpleFactory interface.</td>
</tr>
<tr>
<td>Simples.cpp</td>
<td>The C++ source code for the server application in the Name Service sample application.</td>
</tr>
<tr>
<td>Simplec.cpp</td>
<td>The C++ source code for the client application in the Name Service sample application.</td>
</tr>
<tr>
<td>Simple_i.cpp</td>
<td>The C++ source code that implements the Simple and SimpleFactory methods.</td>
</tr>
<tr>
<td>Simple_i.h</td>
<td>The C++ header file that defines the implementation of the Simple and SimpleFactory methods.</td>
</tr>
</tbody>
</table>
Java Version of the Name Service Sample Application

The files for the Java version of the Name Service sample application are located in the following directories:

Windows NT

\drive:\WLEdir\samples\corba\cnssimpapp_java
Using the Name Service Sample Application

UNIX

/usr/local/WLedir/samples/corba/cnssimpapp_java

You will use the files listed in Table 5-3 to build and run the Java Name Service sample application.

Table 5-3 Files Included in the Java Version of the Name Service Sample Application

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simple.idl</td>
<td>The OMG IDL code that declares the Simple and SimpleFactory interfaces.</td>
</tr>
<tr>
<td>ServerImpl.java</td>
<td>The Java source code that overrides the Server.initialize and Server.release methods.</td>
</tr>
<tr>
<td>SimpleClient.java</td>
<td>The Java source code for the client application in the Name Service sample application.</td>
</tr>
<tr>
<td>SimpleFactoryImpl.java</td>
<td>The Java source code that implements the SimpleFactory methods.</td>
</tr>
<tr>
<td>SimpleImpl.java</td>
<td>The Java source code that implements the Simple methods.</td>
</tr>
<tr>
<td>Simple.xml</td>
<td>The Server Description File used to associate activation and transaction policy values with CORBA interfaces. For the Java version of the Name Service sample application, the Simple and SimpleFactory interfaces have an activation policy of method and a transaction policy of optional.</td>
</tr>
<tr>
<td>Readme.txt</td>
<td>The file that provides the latest information about building and running the Java version of the Name Service sample application.</td>
</tr>
<tr>
<td>runme.cmd</td>
<td>The Windows NT batch file that builds and runs the Java version of the Name Service sample application.</td>
</tr>
</tbody>
</table>
Building and Running the Name Service Sample Application

Step 2: Change the Protection Attribute on the Files for the Name Service Sample Application

During the installation of the WLE software, the sample application files are marked read-only. Before you can edit or build the files in the Name Service sample application, you need to change the protection attribute of the files you copied into your work directory, as follows:

**Windows NT**

prompt>attrib -r drive:\workdirectory\*.*

**UNIX**

prompt>/bin/ksh

Using the CORBA Name Service
5 Using the Name Service Sample Application

```
ksh prompt>chmod u+w /workdirectory/*.*
```

On the UNIX operating system platform, you also need to change the permission of `runme.ksh` to give execute permission to the file, as follows:

```
ksh prompt>chmod +x runme.ksh
```

## Step 3: Verify the Settings of the Environment Variables

Before building and running the Name Service sample application, you need to ensure that certain environment variables are set on your system. In most cases, these environment variables are set as part of the installation procedure. However, you need to check the environment variables to ensure they reflect correct information.

Table 5-4 lists the environment variables required to run the Name Service sample application.

<table>
<thead>
<tr>
<th>Environment Variable</th>
<th>Description</th>
</tr>
</thead>
</table>
| APPDIR               | The directory path where you copied the sample application files. For example:  
  Windows NT  
  APPDIR=c:\work\cnssimpapp  
  UNIX  
  APPDIR=/usr/work/cnssimpapp |
| TUXCONFIG            | The directory path and name of the configuration file. For example:  
  Windows NT  
  TUXCONFIG=c:\work\cnssimpapp\tuxconfig  
  UNIX  
  TUXCONFIG=/usr/work/cnssimpapp/tuxconfig |
Building and Running the Name Service Sample Application

To verify that the information for the environment variables defined during installation is correct, complete the following steps:

**Windows NT**

1. From the Start menu, select Settings.
2. From the Settings menu, select the Control Panel.
   The Control Panel appears.

---

**Table 5-4  Required Environment Variables for the Name Service Sample Application**

<table>
<thead>
<tr>
<th>Environment Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JDKDIR</td>
<td>The directory path where you installed the JDK software. For example:</td>
</tr>
<tr>
<td></td>
<td><strong>Windows NT</strong></td>
</tr>
<tr>
<td></td>
<td>JDKDIR=c:\jdk1.2.2</td>
</tr>
<tr>
<td></td>
<td><strong>UNIX</strong></td>
</tr>
<tr>
<td></td>
<td>JDKDIR=/usr/local/jdk1.2.1</td>
</tr>
<tr>
<td></td>
<td>You need to specify this parameter only if you plan to use the Java version of the Name Service sample application.</td>
</tr>
<tr>
<td>JAVA_HOME</td>
<td>The directory path where you installed the JDK software. For example:</td>
</tr>
<tr>
<td></td>
<td><strong>Windows NT</strong></td>
</tr>
<tr>
<td></td>
<td>JAVA_HOME=c:\JDK1.2</td>
</tr>
<tr>
<td></td>
<td><strong>UNIX</strong></td>
</tr>
<tr>
<td></td>
<td>JAVA_HOME=/usr/local/JDK1.2</td>
</tr>
<tr>
<td></td>
<td>You need to define this environment variable only when you use the Java version of the Name Service sample application.</td>
</tr>
<tr>
<td>RESULTSDIR or JRESULTSDIR</td>
<td>A subdirectory of APPDIR where files that are created as a result of executing the runme command are stored. For example:</td>
</tr>
<tr>
<td></td>
<td><strong>Windows NT</strong></td>
</tr>
<tr>
<td></td>
<td>RESULTSDIR=c:\workdirectory\</td>
</tr>
<tr>
<td></td>
<td><strong>UNIX</strong></td>
</tr>
<tr>
<td></td>
<td>RESULTSDIR=/usr/local/workdirectory/</td>
</tr>
<tr>
<td></td>
<td>When using the Java version of the Name Service sample application, specify the JRESULTSDIR environment variable.</td>
</tr>
</tbody>
</table>
3. Click the System icon.  
   The System Properties window appears.

4. Click the Environment tab.  
   The Environment page appears.

5. Check the settings of the environment variables.

**UNIX**

ksh prompt>printenv TUXDIR
ksh prompt>printenv JAVA_HOME

To change the settings, complete the following steps:

**Windows NT**

1. On the Environment page in the System Properties window, click the environment variable you want to change or enter the name of the environment variable in the **Variable** field.

2. Enter the correct information for the environment variable in the **Value** field.

3. Click OK to save the changes.

**UNIX**

ksh prompt>export TUXDIR=directorypath
ksh prompt>export JAVA_HOME=directorypath

---

**Step 4: Execute the runme Command**

The **runme** command automates the following steps:

1. Setting the system environment variables.

2. Loading the **UBBCONFIG** file.

3. Compiling the code for the client application.

4. Compiling the code for the server application.
5. Starting the server application using the `tmboot` command.

6. Starting the client application.

7. Stopping the server application using the `tmshutdown` command.

Note: You can also run the Name Service sample application manually. The steps for manually running the Name Service sample application are described in the `Readme.txt` file.

To build and run the Name Service sample application, enter the `runme` command, as follows:

**Windows NT**

```
prompt>cd workdirectory
prompt>runme
```

**UNIX**

```
ksh prompt>cd workdirectory
ksh prompt>./runme.ksh
```

The Name Service sample application runs and prints the following messages:

```
Testing simpapp
  cleaned up
  prepared
  built
  loaded ubb
  booted
  ran
  shutdown
  saved results
PASSED
```

Table 5-5 lists the C++ files in the work directory generated by the `runme` command.

**Table 5-5  C++ Files Generated by the runme Command**

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simple_c.cpp</td>
<td>Generated by the <code>idl</code> command, this file contains the client stubs for the SimpleFactory and Simple interfaces.</td>
</tr>
</tbody>
</table>
Table 5-6 lists the Java files in the work directory generated by the runme command.

Table 5-6 Java Files Generated by the runme Command

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SimpleFactory.java</td>
<td>Generated by the m3idltojava command for the SimpleFactory interface. The SimpleFactory interface contains the Java version of the OMG IDL interface. It extends org.omg.CORBA.Object.</td>
</tr>
<tr>
<td>SimpleFactoryHolder.java</td>
<td>Generated by the m3idltojava command for the SimpleFactory interface. This class holds a public instance member of type SimpleFactory. The class provides operations for out and inout arguments that are included in CORBA, but that do not map exactly to Java.</td>
</tr>
<tr>
<td>File</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>SimpleFactoryHelper.java</td>
<td>Generated by the m3idltojava command for the SimpleFactory interface. This class provides auxiliary functionality, notably the narrow method.</td>
</tr>
<tr>
<td>_SimpleFactoryStub.java</td>
<td>Generated by the m3idltojava command for the SimpleFactory interface. This class is the client stub that implements the SimpleFactory.java interface.</td>
</tr>
<tr>
<td>_SimpleFactoryImplBase.java</td>
<td>Generated by the m3idltojava command for the SimpleFactory interface. This abstract class is the server skeleton. It implements the SimpleFactory.java interface. The user-written server class SimpleFactoryImpl extends _SimpleFactoryImplBase.</td>
</tr>
<tr>
<td>Simple.java</td>
<td>Generated by the m3idltojava command for the Simple interface. The Simple interface contains the Java version of the OMG IDL interface. It extends org.omg.CORBA.Object.</td>
</tr>
<tr>
<td>SimpleHolder.java</td>
<td>Generated by the m3idltojava command for the Simple interface. This class holds a public instance member of type Simple. The class provides operations for out and inout arguments that CORBA has but that do not match exactly to Java.</td>
</tr>
<tr>
<td>SimpleHelper.java</td>
<td>Generated by the m3idltojava command for the Simple interface. This class provides auxiliary functionality, notably the narrow method.</td>
</tr>
<tr>
<td>_SimpleStub.java</td>
<td>Generated by the m3idltojava command for the Simple interface. This class is the client stub that implements the Simple.java interface.</td>
</tr>
</tbody>
</table>
Table 5-6 Java Files Generated by the runme Command (Continued)

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_SimpleImplBase.java</td>
<td>Generated by the m3idltojava command for the Simple interface. This abstract class is the server skeleton. It implements the Simple.java interface. The user-written server class SimpleImpl extends _SimpleImplBase.</td>
</tr>
<tr>
<td>Simple.ser</td>
<td>The Server Descriptor File generated by the buildjobjserver command in the runme command.</td>
</tr>
<tr>
<td>Simple.jar</td>
<td>The server Java Archive file generated by the buildjavaserver command in the runme command.</td>
</tr>
<tr>
<td>.adm/.keybd</td>
<td>A file that contains the security encryption key database. The subdirectory is created by the tmloadcf command in the runme command.</td>
</tr>
<tr>
<td>results</td>
<td>A directory generated by the runme command.</td>
</tr>
</tbody>
</table>

Table 5-7 lists files in the RESULTS or JRESULTS directory generated by the runme command.

Table 5-7 Files in the results Directory Generated by the runme Command

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>input</td>
<td>Contains the input that the runme command provides to the Java client application.</td>
</tr>
<tr>
<td>output</td>
<td>Contains the output produced when the runme command executes the Java client application.</td>
</tr>
<tr>
<td>expected_output</td>
<td>Contains the output that is expected when the Java client application is executed by the runme command. The data in the output file is compared to the data in the expected_output file to determine whether or not the test passed or failed.</td>
</tr>
</tbody>
</table>
Table 5-7  Files in the results Directory Generated by the runme Command

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>log</td>
<td>Contains the output generated by the runme command. If the runme command fails, check this file for errors.</td>
</tr>
<tr>
<td>setenv.cmd</td>
<td>Contains the commands to set the environment variables needed to build and run the Java Name Service sample application on the Windows NT operating system platform.</td>
</tr>
<tr>
<td>setenv.ksh</td>
<td>Contains the commands to set the environment variables needed to build and run the Java Name Service sample application on the UNIX operating system platform.</td>
</tr>
<tr>
<td>stderr</td>
<td>Generated by the tmboot command, which is executed by the runme command. If the -noredirect JavaServer option is specified in the UBBCONFIG file, the System.err.println method sends the output to the stderr file instead of to the ULOG file.</td>
</tr>
<tr>
<td>stdout</td>
<td>Generated by the tmboot command, which is executed by the runme command. If the -noredirect JavaServer option is specified in the UBBCONFIG file, the System.out.println method sends the output to the stdout file instead of to the ULOG file.</td>
</tr>
<tr>
<td>tmsysevt.dat</td>
<td>Contains filtering and notification rules used by the TMSYSEVT (system event reporting) process. This file is generated by the tmboot command in the runme command.</td>
</tr>
<tr>
<td>tuxconfig</td>
<td>A binary version of the UBBCONFIG file.</td>
</tr>
<tr>
<td>ubb</td>
<td>The UBBCONFIG file for the Java Name Service sample application.</td>
</tr>
<tr>
<td>ULOG.&lt;date&gt;</td>
<td>A log file that contains messages generated by the tmboot command.</td>
</tr>
</tbody>
</table>
Using the Name Service Sample Application

Run the server application in the Name Service sample application, as follows:

**Windows NT**
prompt> tmboot

**UNIX**
  ksh prompt> tmboot

Run the client application in the Name Service sample application, as follows:

**Windows NT**
prompt> java -classpath %CLIENTCLASSPATH% -DTOBJADDR=%TOBJADDR% SimpleClient
String?
Hello World
hello world

**UNIX**
  ksh prompt> java -classpath $CLIENTCLASSPATH
  /m3envobj.jar -DTOBJADDR=$TOBJADDR SimpleClient
String?
Hello World
hello world

Before using another sample application, enter the following commands to stop the Name Service sample application and to remove unnecessary files from the work directory:

**Windows NT**
prompt> tmshutdown -y
prompt> nmake -f makefile.nt clean

**UNIX**
  ksh prompt> tmshutdown -y
  ksh prompt> make -f makefile.mk clean
Building and Running the Name Service Sample Application
# Index

## A
- administration tasks
  - compressing the persistent storage file 3-4
  - federating the namespace 3-6
  - making the namespace persistent 3-3
  - removing orphan naming context objects 3-5
  - starting the server process 3-2
- AlreadyBound exception
  - described 2-34

## B
- binding iterators
  - defining maximum 2-4
- BindingIterator object
  - described 1-4
  - methods
    - destroy 2-31
    - next_n() 2-32
    - next_one 2-33
- OMG IDL 2-29
  - overview 2-29
- Bootstrap object
  - connecting to the namespace 4-7
  - getting initial references 2-13
  - using the NameService environmental object 4-7

## C
- C++ code examples
  - binding a name to the namespace 4-10
  - connecting to the namespace 4-7
  - locating a name 4-11
  - using a stringified object reference 4-8
- cns command
  - command-line options 2-3
  - compressing the persistent storage file 3-4
  - deleting orphan naming context objects 3-6
  - described 2-3
  - making the namespace persistent 3-3
  - syntax 2-3
- cnbind command
  - binding objects to the namespace 2-6
  - command-line options 2-6
  - described 2-6
  - examples 2-7
  - syntax 2-6
- cnsls command
  - command-line options 2-9
  - described 2-9
  - displaying the contents of the namespace 2-9
  - example 2-10
  - syntax 2-9
cnsunbind command
  command-line options 2-11
  deleting bindings from the namespace 2-11
described 2-11
  examples 2-12
  syntax 2-11
commands
cns 2-3
cnsbind 2-6
cnsls 2-9
cnsunbind 2-11
CosNaming data structures
  BindingList 2-13
  BindingType 2-13
  Istring 2-13
  listed 2-13
  Name 2-14
  NameComponent 2-14
CosNaming interfaces
  compiling the OMG IDL 4-6
  directory location of OMG IDL 4-3
  obtaining the OMG IDL 4-3
customer support contact information ix

D
dangling bindings
  defined 2-3
  deleting 2-3, 3-5
directory location of source files
  Name Service sample application 5-4
  OMG IDL for CosNaming interfaces 4-3
documentation, where to find it viii

E
environment variables
  APPDIR 5-9
  JAVA_HOME 5-8
  JDKDIR 5-9
Name Service sample application 5-8
  TOBJADDR 5-9
  TUXCONFIG 5-9
  TUXDIR 5-8
exceptions
  AlreadyBound 2-34
  InvalidAddress 2-35
  InvalidName 2-36
  NotEmpty 2-37
  NotFound 2-38

F
federation
  inbound 3-6
  outbound 3-7
  the ISL command 3-7
file protections
  Name Service sample application 5-7

I
IIOP Listener/Handler
  enabling
    inbound federation 3-6
    outbound federation 3-7
installation
  requirements 1-2
InvalidAddress exception
  defined 2-35
InvalidName exception
  defined 2-36
ISL command
  inbound federations 3-7
  outbound federation 3-7

J
Java code examples
  binding a name to the namespace 4-10
  connecting to the namespace 4-7
Using the WebLogic Enterprise CORBA Name Service

I-3

locating a name 4-12
using a stringified object reference 4-8
JAVA_HOME parameter
Name Service sample application 5-8
JDKDIR parameter
setenv file 5-9

M
makefile
Name Service sample application 5-7

N
described 1-3
name
locating in the namespace 4-11
Name Service sample application
building 5-3
changing protection on files 5-7
compiling
the C++ client application 5-11
the C++ server application 5-11
the Java client application 5-10
the Java server application 5-10
loading the UBBCONFIG file 5-11
locations of files 5-4
required environment variables 5-8
runme command 5-10
setting up the work directory 5-4
source files 5-4
starting the Java client application 5-16
starting the Java server application 5-16
using the client applications 5-16
NameService environmental object
connecting to the namespace 4-7
described 2-13
using the Bootstrap object 2-13
namespace
binding an object to 4-9
cnsls command 2-9
cnsunbind command 2-11
connecting
using a stringified object reference 4-7
with the Bootstrap object 4-7
deleting bindings from 2-11
displaying the contents 2-9
federating 3-6
making persistent 3-3
NamingContext object
described 1-3
methods
bind 2-15
bind_context 2-16
bind_new_context 2-17
destroy 2-18
list 2-19
new_context 2-20
rebind 2-21
rebind_context 2-22
resolve 2-23
unbind 2-24
OMG IDL 2-14
overview 2-14
NamingContextExt object
described 1-4
methods
resolve_str 2-26
to_name 2-27
to_string 2-28
to_URL 2-29
OMG IDL 2-24
overview 2-24
NotEmpty exception
defined 2-37
NotFound exception
defined 2-38
Using the WebLogic Enterprise CORBA Name Service

OMG IDL
  BindingIterator object 2-29
  compiling 4-6
  file name 4-3
  for NamingContext object 2-14
  location on the kit 4-3
  NamingContextExt object 2-24
  Simple interface 5-2
  SimpleFactory interface 5-2
orphan naming context objects
  creating 3-5
  defined 2-3
  deleting 3-5

persistent storage file
  compressing 3-4
  creating 3-4
printing product documentation viii
programming tasks
  binding objects to the namespace 4-9
  compiling the OMG IDL 4-6
  connecting to the namespace 4-7
  obtaining the OMG IDL 4-3
  overview 4-2
  using a name to locate an object 4-11

related information viii
resolve method
  overview 1-4
runme command
  description 5-10
  files generated by 5-11, 5-12

source files
  Name Service sample application 5-6
stringified object references
  connecting to the namespace 4-7
  restrictions for 4-9
support
  technical ix

tmboot command
  Name Service sample application 5-16
tmloadcf command
  Name Service sample application 5-10
TOBJADDR environment variable
  cnbind command 2-6
  use with cnsls command 2-9
  use with cnsubind command 2-11
TUXCONFIG parameter
  setenv file 5-9
TUXDIR parameter
  Name Service sample application 5-8
UBBCONFIG file
  example
    for inbound federation 3-7
    for Name Server server process 3-2
    Name Service sample application 5-11
ULOG file
  persistent storage file 3-4

WLE CORBA Name Service
  capabilities 2-12
  commands 2-1
  CosNaming data structures 2-13
  exceptions 2-33
features 1-1
illustrated 1-2
installing 3-1
limitations 2-12
overview 1-1
requirements 1-2