Federated Naming Service Guide
## Contents

**Part 1 — Introduction**

1. **Introduction to the Federated Naming Service (FNS)**  
   - What Is FNS? .................................................. 4  
   - What Is XFN? .................................................. 5  
   - Why FNS? ...................................................... 5  
      - Uniform Naming Interface ............................... 5  
      - Uniform Means of Composing Names .................. 6  
      - Coherence in Naming .................................... 7  
   - FNS in the Solaris Environment ........................... 8  
      - FNS and NIS+ ............................................ 8  
      - FNS and DNS ............................................ 9  
      - FNS and X.500 .......................................... 9  
      - FNS-based File Naming ................................ 9  
      - FNS-based Printer Naming ............................. 9  
      - FNS and Applications .................................. 10
2. The XFN Model ........................................ 11
   XFN Architectural Model .............................. 11
   References ........................................... 11
   Contexts ............................................. 12
   Attributes ......................................... 13
   Compound Names ................................. 13
   Composite Names .................................. 15
   XFN Links ........................................... 15
   Initial Context ..................................... 16
   User’s View ......................................... 17
   File System View ...................................... 18
   Application View .................................... 19
   API Usage Model ..................................... 22

Part 2 — FNS Policies

3. Introduction to FNS Policies ......................... 25
   Policy Overview .................................... 26
   What FNS Policies Specify ...................... 26
   What FNS Policies Do Not Specify .............. 26
   What FNS Enterprise Policies Arrange ........... 28
   Examples of Composite Names .................... 29
   How FNS Policies Relate to NIS+ ............... 30
   NIS+ Domains and FNS Organizational Units .... 30
   NIS+ Users and FNS Users ....................... 31
   NIS+ Hosts and FNS Hosts ....................... 32
Target Client Applications of FNS Policies ....................... 32
Example Application: Calendar Service ....................... 33

4. Policies for the Enterprise Namespace .................... 37
   Namespaces in the Enterprise ......................... 37
      Organizational Unit Namespace .................... 38
      Site Namespace ................................. 38
      User Namespace ............................... 39
      Host Namespace ............................... 39
      Service Namespace ............................ 39
      Printer Namespace ............................ 40
      File Namespace ............................... 40
      Namespace Identifiers ......................... 41
   Structure of the Enterprise Namespace .................... 42
      Enterprise Root ............................... 45
      Organizational Units ......................... 46
      Sites ......................................... 47
      Users ......................................... 49
      Hosts ......................................... 50
      Services ....................................... 51
      Files .......................................... 52
      Printers ....................................... 53
   Initial Context Bindings for Naming Within the Enterprise 54
      User-related Bindings ......................... 56
      Host-related Bindings ......................... 58
"Shorthand" Bindings .............................. 59
5. Policies for the Global Namespace ..................... 61
   The Global Namespace ............................. 61
   Initial Context Bindings for Global Naming .......... 62
   Federating DNS .................................. 62
   Federating X.500 ................................ 63

Part 3 —Administration

6. Administering FNS on NIS+ .......................... 69
   Setting Up FNS .................................... 70
      Estimating Resource Requirements ................ 70
   Setting Up NIS+ Service for FNS ..................... 70
   Setting Up the FNS Namespace ....................... 71
   Replicating FNS Service ............................ 72
   Creating FNS Contexts Individually .................. 73
      Organization Context ............................ 75
      All Hosts Context ............................... 76
      Single Host Context ............................. 76
      All–Users Context ............................... 77
      Single User Context .............................. 78
      Service Context ................................. 79
      Printer Context ................................. 80
      Generic Context ................................. 80
      Site Context ..................................... 81
      File Context ..................................... 82
Troubleshooting ....................................................... 104
  Cannot Obtain Initial Context ................................. 104
  Nothing in Initial Context .................................. 105
  “No Permission” Messages .................................. 105
  fnlist Does Not List Suborganizations ..................... 106
  Cannot Create Host- or User-related Contexts .......... 107
  Cannot Remove a Context I Created ....................... 107
  “Name in Use” With fnunbind .............................. 108
  “Name in Use” With fnbind/fncreate -s ................. 108
  fndestroy/fnunbind Does Not Return “Operation Failed” ....................................................... 109

7. Federating NIS+ With Global Naming Systems ........... 111
   Obtaining the NIS+ Root Reference ....................... 111
   Federating NIS+ Under DNS ............................... 112
   Federating NIS+ Under X.500 ............................. 114

8. Administering the File System Namespace ............... 117
   The FNS File System Namespace ......................... 117
   NFS File Servers ........................................... 118
   The Automounter ........................................... 119
   Creating File Contexts ................................... 120
   Creating the Input File ................................... 121
   Using Command-line Input ................................ 123
   Advanced Input Formats ................................. 124
   Backward Compatibility Input Format .................... 125
Administering File Contexts ........................................... 126

9. Administering the Printer Namespace ....................... 127
   The Printer Namespace ........................................... 127
   Administering Printer Contexts ............................... 128
      Using Files .................................................... 128
      Using NIS ..................................................... 129
      Using NIS+ ................................................... 129

Part 4 — Application Programming

10. Interfaces for Writing XFN Applications ................... 133
    XFN Interface Overview ........................................ 133
    Interface Conventions .......................................... 134
    Usage ............................................................ 134
    Abstract Data Types ........................................... 134
    Memory–Management Policies ................................. 135
    The Base Context Interface .................................... 135
    Names in Context Operations ................................. 136
    Requirements for Supporting the Context Operations .... 136
    Status Objects .................................................. 137
    Getting Context Handles ...................................... 137
    Lookup and List Contexts ...................................... 138
    Updating Bindings .............................................. 140
    Managing Contexts ............................................. 141
    Other Context Operations .................................... 142
    The Base Attribute Interface .................................. 143
### Federated Naming Service Guide

**XFN Attribute Model** ........................................ 143
**Relationship to Naming Operations** .................... 144
**Status Objects** ............................................. 145
**Single-Attribute Operations** .............................. 145
**Multiple-Attribute Operations** ........................... 147
**Status Objects and Status Codes** ........................ 150
**Parameters Used in the Interface** ........................ 153
**Composite Names** .......................................... 153
**References and Addresses** ................................ 153
**Identifiers** .................................................. 154
**Strings** ...................................................... 155
**Attributes and Attribute Values** ......................... 155
**Attribute Sets** ............................................ 155
**Attribute-Modification Lists** ............................. 155
**Parsing Compound Names** .................................. 156
**Syntax Attributes** ......................................... 156
**XFN Standard Syntax Model** .............................. 156

### 11. XFN Composite Names ................................. 159

**Syntax** ..................................................... 159
**Composite Name and Naming System Boundaries** ...... 161
**Strong Separation** ......................................... 161
**Weak Separation** .......................................... 162
**Composite Name Resolution** .............................. 163
**Explicit NNSPs: Junctions** ................................. 163
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Implicit NNSPs</td>
<td>164</td>
</tr>
<tr>
<td>Coexistence of Explicit and Implicit NNSPs</td>
<td>165</td>
</tr>
<tr>
<td>XFN Links</td>
<td>165</td>
</tr>
<tr>
<td>12. XFN Programming Examples</td>
<td>167</td>
</tr>
<tr>
<td>Namespace Browser Example</td>
<td>168</td>
</tr>
<tr>
<td>Compiling and Executing Browser Example</td>
<td>175</td>
</tr>
<tr>
<td>Commands</td>
<td>175</td>
</tr>
<tr>
<td>Sample Output</td>
<td>176</td>
</tr>
<tr>
<td>Printer Programming Example</td>
<td>178</td>
</tr>
<tr>
<td>Client</td>
<td>179</td>
</tr>
<tr>
<td>Server</td>
<td>181</td>
</tr>
<tr>
<td>A. XFN Composite Names Syntax</td>
<td>185</td>
</tr>
<tr>
<td>Composite Name Encoding</td>
<td>185</td>
</tr>
<tr>
<td>Backus-Naur Form (BNF)</td>
<td>186</td>
</tr>
<tr>
<td>Decomposing the Composite Name String</td>
<td>187</td>
</tr>
<tr>
<td>Composing the Composite Name String</td>
<td>189</td>
</tr>
<tr>
<td>B. DNS Text Record Format for XFN References</td>
<td>191</td>
</tr>
<tr>
<td>C. X.500 Attribute Syntax for XFN References</td>
<td>195</td>
</tr>
<tr>
<td>Object Classes</td>
<td>195</td>
</tr>
<tr>
<td>Glossary</td>
<td>201</td>
</tr>
<tr>
<td>Index</td>
<td>207</td>
</tr>
</tbody>
</table>
## Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 4-1</td>
<td>Namespace Identifiers in the Enterprise</td>
<td>41</td>
</tr>
<tr>
<td>Table 4-2</td>
<td>Policies for the Federated Enterprise Namespace</td>
<td>43</td>
</tr>
<tr>
<td>Table 4-3</td>
<td>Initial Context Bindings for Naming Within the Enterprise</td>
<td>55</td>
</tr>
<tr>
<td>Table 5-1</td>
<td>Policies for the Federated Global Namespace</td>
<td>61</td>
</tr>
<tr>
<td>Table 5-2</td>
<td>Initial Context Bindings for Global Naming</td>
<td>62</td>
</tr>
<tr>
<td>Table 6-1</td>
<td>fncreate Command Options</td>
<td>74</td>
</tr>
<tr>
<td>Table 6-2</td>
<td>fnlookup Command Options</td>
<td>83</td>
</tr>
<tr>
<td>Table 6-3</td>
<td>fnlist Command Options</td>
<td>85</td>
</tr>
<tr>
<td>Table 6-4</td>
<td>fnbind Command Options</td>
<td>89</td>
</tr>
<tr>
<td>Table 6-5</td>
<td>fncheck Command Options</td>
<td>95</td>
</tr>
<tr>
<td>Table 7-1</td>
<td>NIS+ Root Reference</td>
<td>112</td>
</tr>
<tr>
<td>Table 8-1</td>
<td>fncreate_fs Command Options</td>
<td>121</td>
</tr>
<tr>
<td>Table 10-1</td>
<td>XFN Attribute -Modification Operations</td>
<td>146</td>
</tr>
<tr>
<td>Table 10-2</td>
<td>Status Object</td>
<td>150</td>
</tr>
<tr>
<td>Table 10-3</td>
<td>Status Codes</td>
<td>151</td>
</tr>
<tr>
<td>Table 10-4</td>
<td>XFN Identifier Formats</td>
<td>154</td>
</tr>
<tr>
<td>Table</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>---------------</td>
<td>-------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>Table 10-5</td>
<td>XFN Syntax Attributes</td>
<td>158</td>
</tr>
<tr>
<td>Table 11-1</td>
<td>String and Structural Forms of XFN Composite Names</td>
<td>160</td>
</tr>
<tr>
<td>Table 12-1</td>
<td>Namespace Browser Commands</td>
<td>175</td>
</tr>
<tr>
<td>Table 12-2</td>
<td>Backus-Naur Notation</td>
<td>186</td>
</tr>
<tr>
<td>Table 12-3</td>
<td>XFN Composite Name Syntax Using BNF</td>
<td>186</td>
</tr>
</tbody>
</table>
Figures

Figure 1-1  Naming Is an Integral Part of Existing Services .............. 4
Figure 1-2  A Federated Naming System ................................. 7
Figure 2-1  An XFN Context ............................................... 12
Figure 2-2  Hierarchical Naming System With Compound Names ...... 14
Figure 2-3  Federated Naming System With Composite Names .......... 16
Figure 2-4  User View of XFN ............................................ 17
Figure 2-5  User Interaction With XFN ................................. 18
Figure 2-6  Client Application Interaction With XFN .................... 19
Figure 2-7  Details Beneath XFN API ................................. 20
Figure 2-8  XFN Implementation Examples ............................. 21
Figure 3-1  Different Levels of Name Services .......................... 27
Figure 3-2  What FNS Policies Arrange ................................. 28
Figure 4-1  Example of an Enterprise Namespace ..................... 44
Figure 4-2  Example of Enterprise Bindings in the Initial Context ...... 54
Figure 5-1  Example of an X.500 Directory Information Base .......... 64
Figure 8-1  NFS File System—Simple Case ............................. 118
Preface

The Federated Naming Service (FNS) is new to the Solaris™ product family. FNS is a set of application programming interfaces and policies that allow applications to use a common set of names and policies over different name services.

FNS is not a replacement for NIS+, the network name service included in the Solaris software environment. Rather, FNS is implemented on top of NIS+ and allows you to use a set of common names with desktop applications. SunSoft Inc.’s implementation of FNS conforms to the X/Open™ federated naming (XFN) specification.

Who Should Use This Book

The primary audience of Federated Naming Service Guide is software developers who write distributed applications. Use of this guide assumes basic competence in programming, a working familiarity with the C programming language, and a working familiarity with the UNIX® operating system. Developers should read all four parts of this manual.

The secondary audiences are system and network administrators and application users. All should read Part 1 to get an overview of FNS. Administrators should also read Part 2 and, especially, Part 3 to set up and administer FNS. This manual does not cover NIS+ or the Domain Name System (DNS) except as they relate to FNS.
How This Book Is Organized

Part 1—Introduction
Chapter 1, “Introduction to the Federated Naming Service (FNS),” is a high-level overview of what FNS is and the problems it addresses.

Chapter 2, “The XFN Model,” depicts the architectural model of federated naming from the application, API, and end-user’s views.

Part 2—FNS Policies
Chapter 3, “Introduction to FNS Policies,” introduces FNS enterprise and global policies.

Chapter 4, “Policies for the Enterprise Namespace,” explains the policies for naming objects within an enterprise and how applications can use these policies.

Chapter 5, “Policies for the Global Namespace,” describes naming objects in global namespaces.

Part 3—Administration
Chapter 6, “Administering FNS on NIS+,” is a reference for system administrators who need to administer FNS in an NIS+ environment.

Chapter 7, “Federating NIS+ With Global Naming Systems,” describes the procedures for federating NIS+ with DNS and X.500.

Chapter 8, “Administering the File System Namespace,” describes the setup and administration of the file system namespace.

Chapter 9, “Administering the Printer Namespace,” describes the setup and administration of the printer namespace.

Part 4—Application Programming
Chapter 10, “Interfaces for Writing XFN Applications,” defines the client programming interfaces.

Chapter 11, “XFN Composite Names,” describes the XFN composite name string syntax and the resolution techniques for composite names.
Chapter 12, “XFN Programming Examples,” presents self-contained executable programs for a namespace browser and a printer client and server.

Appendixes

Appendix A, “XFN Composite Names Syntax,” gives supplemental information about composite name syntax.

Appendix B, “DNS Text Record Format for XFN References,” gives supplemental information about FNS in a DNS environment.

Appendix C, “X.500 Attribute Syntax for XFN References,” gives supplemental information about FNS in a X.500 environment.

Related Books

With the exception of the XFN specification, these books do not specifically cover FNS but they provide a good background on how name services work in client-server computing:

- *Distributed Computing—Implementation and Strategy* by Raman Khanna (Prentice Hall, 1993)
- *DNS and BIND* by P. Albitz and C. Liu (O’Reilly, 1992)
- *Managing the X.500 Client Toolkit* (SunSoft Inc., 1995)

You may also want to reference the following AnswerBook® on-line documentation:

- *Solaris 2.5 Reference Manual AnswerBook*
- *Solaris 2.5 Software Developer AnswerBook*
- *Solaris 2.5 System Administrator AnswerBook*
What Typographic Changes and Symbols Mean

The following table describes the typographic changes used in this book.

Table P-1  Typographic Conventions

<table>
<thead>
<tr>
<th>Typeface or Symbol</th>
<th>Meaning</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>AaBbCc123</td>
<td>The names of commands, files, and directories; on-screen computer output</td>
<td>Edit your .login file. Use ls -a to list all files. machine_name% You have mail.</td>
</tr>
<tr>
<td>AaBbCc123</td>
<td>What you type, contrasted with on-screen computer output</td>
<td>machine_name% su Password:</td>
</tr>
<tr>
<td>AaBbCc123</td>
<td>Command-line placeholder: replace with a real name or value</td>
<td>To delete a file, type rm filename.</td>
</tr>
<tr>
<td>AaBbCc123</td>
<td>Book titles, new words or terms, or words to be emphasized</td>
<td>Read Chapter 6 in User’s Guide. These are called class options. You must be root to do this.</td>
</tr>
</tbody>
</table>

Shell Prompts in Command Examples

The following table shows the default system prompt and superuser prompt for the C shell, Bourne shell, and Korn shell.

Table P-2  Shell Prompts

<table>
<thead>
<tr>
<th>Shell</th>
<th>Prompt</th>
</tr>
</thead>
<tbody>
<tr>
<td>C shell prompt</td>
<td>machine_name%</td>
</tr>
<tr>
<td>C shell superuser prompt</td>
<td>machine_name#</td>
</tr>
<tr>
<td>Bourne shell and Korn shell</td>
<td>$</td>
</tr>
<tr>
<td>Bourne shell and Korn shell</td>
<td>#</td>
</tr>
</tbody>
</table>
Part 1 — Introduction

This part of the manual is an overview of FNS and its naming model.

<table>
<thead>
<tr>
<th>Introduction to the Federated Naming Service (FNS)</th>
<th>page 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>The XFN Model</td>
<td>page 11</td>
</tr>
</tbody>
</table>
Introduction to the Federated Naming Service (FNS)

This chapter is a high-level overview of the Federated Naming Service (FNS).

Name services are fundamental to any computing system. Among other features, a name service provides functionality that

- Associates names with objects (binds)
- Resolves names to objects
- Removes bindings
- Lists names
- Renames

Name services are often embedded in many different applications and services in a computing environment, as shown in Figure 1-1 on page 4. Working with different name services presents significant difficulties to the application developer. Most applications are designed to use a single name service and have very limited access to objects in a distributed computing environment. Moreover, different applications use different name services and expect names to be composed differently. They often use different names for what the user considers very similar objects. For example, you may be able to send mail to
your friend Joan using her name joan@admin, but be required to use another name, jsmith@hal, to access her calendar. FNS allows the user to name objects such as users in a uniform way.

What Is FNS?

FNS provides a method for federating multiple name services under a single, simple uniform interface for the basic naming operations. The service supports resolution of composite names—names that span multiple name systems—through the naming interface. Each member of a federation has autonomy in its choice of naming conventions, administrative interfaces, and its particular set of operations other than name resolution.

In the Solaris environment, the FNS implementation consists of a set of enterprise-level name services with specific policies and conventions for naming organizations, users, hosts, sites, and services as well as support for global name services such as DNS and X.500.
What Is XFN?

XFN is X/Open Federated Naming. XFN is a standard actively supported by organizations such as SunSoft, Inc., IBM, Hewlett-Packard, DEC, Siemens, and OSF. FNS, the Solaris implementation, is compliant with the *X/Open Preliminary Specification for Federated Naming* (July 1994). Applications that use FNS are portable across platforms because the interface exported by FNS is XFN, a public, open interface endorsed by other vendors and X/Open. The X/Open Co. Ltd. is an international standards organization committed to defining computing standards that are endorsed and adhered to by the major computer vendors.

Note – In this manual it is important to distinguish between XFN and FNS. For example, the FNS policies include some extensions to XFN policies, and these are explicitly defined with notes. Objects belonging to the XFN programming interface are designated as XFN objects in order to avoid confusion with other programming interfaces.

Why FNS?

FNS is useful for the following reasons:

- A single uniform naming interface is provided to clients for accessing different name services. As a consequence, the addition of new name services does not require changes to applications or to existing member name services.
- Names can be composed in a uniform way, and the resulting composite names can have any number of components.
- Coherent naming is encouraged through the use of shared contexts and shared names.

The following subsections expand on these reasons.

Uniform Naming Interface

Within the Solaris environment, name services are integrated into other services such as the file system, the network information service, the mail system, and the calendar service. For example, the file system includes a
naming system for files and directories; NIS+ service combines a naming system with a specialized information service; a spreadsheet application names cells and macros. An application in the Solaris environment must often deal with this diversity of name service interfaces. In addition, the application may be exposed to a great variety of often incompatible naming systems external to the Solaris environment. Local- and wide-area networks connect a heterogeneous array of hardware and operating systems, increasing the variety of potential interfaces. Not only do these naming interfaces differ widely, but the essential naming operations are often obscure.

A standard interface that provides the basic naming functions greatly simplifies the naming aspect of applications for developers because now a single interface to name different types of objects can be used. Changes to the underlying name service and adding new name services can be applied without requiring changes to the applications.

**Uniform Means of Composing Names**

Historically, a small percentage of applications use composite names to access objects in the Solaris environment. The commands `mail` and `rcp` are examples of such applications. `rcp` uses composite names such as `sylvan:/usr/jsmith/memo`, which has two components: the host name `sylvan` and the file name (path) `/usr/jsmith/memo`. The `mail` program uses composite names such as `jsmith@hal`, which has two components: the user name `jsmith` and the host name `hal`. Each application defines its own composition rule for names, parses the composite names, and resolves composite names. Composition rules often differ from one application to another.

The user must remember which applications permit composite naming and which do not. For example, the composite name `sylvan:/tmp/foo` is accepted by the `rcp` command, but not by the `cp` command. The user must also remember the different composition rules used among different applications. Applications that support composite names on their own can use only a small and specific set of naming systems, and must be changed whenever a new type of naming system is added.

Incorporating a uniform policy for composite naming into the computing platform permits any application to support composite names in a uniform way.
Figure 1-2 shows an example of a federated naming system consisting of two component naming systems: a host naming system and a file naming system. When the name \texttt{jurassic/usr/games/bin/pirates.exe} is passed to the federated naming system, the host part of the name, \texttt{jurassic}, is resolved by the host naming system, while the file part of the name, \texttt{/usr/games/bin/pirates.exe}, is resolved by the file naming system. The important point to note is that the application passes \textit{one} name to \textit{one} interface.

![Federated Naming System Diagram](image)

\textbf{Figure 1-2}  \hspace{1cm} A Federated Naming System

\section*{Coherence in Naming}

At present, users of the Solaris environment must use different, inconsistent names to refer to objects. To expand on an earlier example: you may use the name \texttt{jsmith@admin} to send mail to Joan, the name \texttt{jsmith@hal} to access her calendar, and the name \texttt{/home/jsmith/.cshrc} to reach a file in her home directory. This disparity makes it hard for users to formulate names and hard for applications to automatically generate names on behalf of users. FNS policies define a coherent way for naming these objects.

The following principles were used to arrive at FNS policies:

\begin{itemize}
  \item \textit{When it is natural to name other objects relative to a certain object, that object should provide a naming context.} For example, because it is natural to want to name various things relative to a user, a user object should be a naming context.
\end{itemize}
• It should be possible to compose names using common components. This reduces the number of names that users need to remember and makes it easier for applications and users to construct names based on their knowledge of common constituents and how they can be logically composed.

• Names should be intuitive and self-evident. For example, the calendar name jsmith@hal names the host where the calendar service is being provided. To the user, there is no obvious connection between the user’s calendar and a host. The host name is extraneous and difficult to discover and remember.

• Never use two contexts when one context will do. In the example above, we would like to name a mail address, a calendar, and a file’s directory relative to the user jsmith. Sharing contexts and their names make naming more coherent and simplifies administration.

**FNS in the Solaris Environment**

In the Solaris environment, the FNS implementation currently consists of a set of enterprise-level name services implemented on top of NIS+, global-level naming systems using DNS and X.500, file naming, and support for printer naming. FNS will become increasingly more visible to the Solaris user as more applications and systems use FNS.

**FNS and NIS+**

NIS+ is the enterprise-wide information service in the Solaris environment. It is an information-retrieval system for well-known UNIX databases, such as the password tables, host tables, and mail aliases tables. It also supports Solaris-specific databases such as the automount maps and the credentials tables. It partitions an enterprise into organizational units that are arranged into a tree and assigned hierarchical domain names.

FNS federates NIS+ in order to support enterprise-level naming policies in the Solaris environment. To do this, FNS provides the XFN interface for performing naming operations on organization, site, user, and host objects. It implements these operations using the NIS+ programming interface for accessing NIS+ directories and tables. It stores bindings for enterprise-level objects in NIS+ and uses them in conjunction with the standard NIS+ tables for passwords and hosts.
**FNS and DNS**

The Internet Domain Name System (DNS) is a hierarchical collection of name servers that provide the Internet community with host and domain name resolution. FNS uses DNS to name entities globally. Names can be constructed for any enterprise that is accessible on the Internet; consequently, names can also be constructed for objects exported by these enterprises. For more information about FNS and DNS, see “Federating DNS” on page 62.

**FNS and X.500**

X.500 is a global directory service. Its components cooperate to manage information about objects in a worldwide scope. Such objects include countries, organizations, people, and machines. FNS federates X.500 in order to enable global access to enterprise name services. For more information about FNS and X.500, see “Federating X.500” on page 63.

**FNS-based File Naming**

FNS-based file naming integrates FNS naming into the Solaris file service. FNS-based file naming enables files to be named relative to users, hosts, sites, and organizations, using the FNS policies shared with other non-file applications. FNS-based file naming gives clients a common view of the global and enterprise-wide file namespaces. Solaris applications that access the file system will, without modification, have access to the file namespaces supported by FNS.

**FNS-based Printer Naming**

FNS-based printer naming provides the basic naming support for the unbundled SunSoft Print Client (SSPC). FNS-based printer naming enables printers to be named relative to users, hosts, sites, and organizations, using the FNS policies shared with other non-printing-related applications. FNS-based printer naming gives clients a common view of the global and enterprise-wide printer namespaces and allows centralized administration of the printer namespaces.
FNS and Applications

Applications that are aware of FNS can expect the namespace to be arranged according to the FNS policies, and applications that bind names in the FNS namespace are expected to follow these policies.

There are three ways for applications to use FNS:

• **Applications could be direct clients of the XFN interface and policies.** Application-level utilities such as the file system, the printing service, and the desktop tools (calendar manager, file manager) are examples of clients that use the XFN interface directly.

• **Applications can use FNS through existing interfaces.** A significant proportion of FNS use will be through existing application programming interfaces. For example, consider a UNIX application that obtains a file name that it later supplies to the UNIX `open()` function. With FNS support for resolution of file names, the application need not be aware that the strings it deals with are composite names rather than the traditional local path names. Many applications can thereby support the use of composite names without modification.

• **Systems can export the XFN interface.** Naming systems, such as DNS and X.500, and naming systems embedded in other services, like the file system and printing service, are examples of naming systems that export the XFN interface.
The XFN Model

This chapter describes the XFN naming model from several perspectives. Some aspects are graphically presented as a way of helping you visualize XFN concepts.

<table>
<thead>
<tr>
<th>XFN Architectural Model</th>
<th>page 11</th>
</tr>
</thead>
<tbody>
<tr>
<td>User’s View</td>
<td>page 17</td>
</tr>
<tr>
<td>Application View</td>
<td>page 19</td>
</tr>
<tr>
<td>API Usage Model</td>
<td>page 22</td>
</tr>
</tbody>
</table>

XFN Architectural Model

The primary services provided by a federated naming system are mapping a composite name to a reference and providing access to attributes associated with a named object. This section defines the elements of the XFN naming model.

References

A reference is the information on how to reach an object. It contains a list of addresses. An address identifies a communication endpoint. Multiple addresses identify multiple communication endpoints for a single conceptual object or service. For example, a list of addresses may be required because the object is distributed or because the object can be accessed through more than one communication mechanism.
**Note** – XFN cannot guarantee specific properties of addresses such as their stability, validity, or reachability. A client may be able to look up a name but not be able to use the returned reference because the client may not have support for any of the necessary communication mechanisms or may lack the necessary network connectivity to reach the address. Further, the address may be invalid from that origin or stale; these issues are the province of convention between the name’s binder, the clients, and the service provider specified in the address.

## Contexts

An XFN context is an object that exports the XFN base context programming interface. A context contains a list of atomic names bound to references, as shown in Figure 2-1. An atomic name may have zero or more attributes. Contexts are at the heart of the lookup and binding operations, described extensively in Chapter 10, “Interfaces for Writing XFN Applications.”

![Figure 2-1 An XFN Context](image)
Attributes

Each named object is associated with zero or more attributes. Thus, attributes are optional, as shown by the dotted lines in Figure 2-1 on page 12. Each attribute has a unique attribute identifier, an attribute syntax, and a set of zero or more distinct attribute values. XFN defines the base attribute interface for examining and modifying the values of attributes associated with existing named objects. These objects may be contexts or any other types of objects. Associated with a context are syntax attributes that describe how the context parses compound names.

Compound Names

A compound name is a sequence of one or more atomic names. An atomic name in one context object can be bound to a reference to another context object of the same type, called a subcontext. Objects in the subcontext are named using a compound name. Compound names are resolved by looking up each successive atomic name in each successive context.

A familiar analogy for UNIX users is the file naming model, where directories are analogous to contexts, and path names serve as compound names. Furthermore, contexts can be arranged in a “tree” structure, just as directories are, with the compound names forming a hierarchical namespace.

- UNIX example: `usr/local/bin`. UNIX atomic names are ordered from left to right and are delimited by slash (`/`) characters. The name `usr` is bound to a context in which `local` is bound. The name `local` is bound to a context in which `bin` is bound.

- DNS example: `sales.Wiz.COM`. DNS atomic names are ordered from right to left, and are delimited by dot (`.`) characters. The domain name `COM` is bound to a context in which `Wiz` is bound. `Wiz` is bound to a context in which `sales` is bound.

- X.500 example: `c=us/o=wiz/ou=sales`. An X.500 atomic name comprises an attribute type and an attribute value. Atomic names are known as relative distinguished names in X.500. In this string representation X.500 atomic names are ordered from left to right, and are delimited by slash (`/`) characters. An attribute type is separated from an attribute value by an equal sign (`=`) character. Abbreviations are defined for commonly used attribute types (for
example, “c” represents country name). The country name US is bound to a context in which wiz is bound. The organization name wiz is bound to a context in which the organizational unit name sales is bound.

Figure 2-2 shows an example of a hierarchical naming system with compound names.

![Hierarchical Naming System With Compound Names](image)

**Figure 2-2** Hierarchical Naming System With Compound Names
Composite Names

A composite name is a name that spans multiple naming systems. It consists of an ordered list of zero or more components. Each component is a name from the namespace of a single naming system. Composite name resolution is the process of resolving a name that spans multiple naming systems. Chapter 11, “XFN Composite Names,” and Appendix A, “XFN Composite Names Syntax,” supply more detail about composite names.

Components are slash-separated (/) and ordered from left to right, according to XFN composite name syntax. For example, the composite name

```
sales.Wiz.COM/usr/local/bin
```

has two components, a DNS name (sales.Wiz.COM) and a UNIX path name (usr/local/bin).

Figure 2-3 on page 16 shows an example of a federated naming system with composite names. Note that the position of the name within a context has no inherent significance in this illustration.

XFN Links

An XFN link is a special form of a reference that is bound to an atomic name in a context. Instead of an address, a link contains a composite name. Many naming systems support a native notion of link that may be used within the naming system itself. XFN does not specify whether there is any relationship between such native links and XFN links.

“XFN Links” on page 165 describes links in detail.
Figure 2-3  Federated Naming System With Composite Names

Initial Context

Every XFN name is interpreted relative to some context, and every XFN naming operation is performed on a context object. The initial context object provides a starting point for the resolution of composite names. The XFN interface provides a function that allows the client to obtain an initial context.

The policies described in Part 2 of this guide specify a set of names that the client can expect to find in this context and the semantics of their bindings. This provides the initial pathway to other XFN contexts.
User’s View

Users experience federated naming through applications, as shown in Figure 2-4. The user can interact with XFN-aware applications in a simple, intuitive, and consistent manner. Typically, the user does not need to compose or know the full composite name of the objects because the application takes care of constructing the composite names.

For example, if the application is expecting you to type a user name, the application may include the user/ string in front of names that you enter. Furthermore, if the application needs to name one of the user’s services, such as the user’s default fax machine, it can append the rest of the name, /service/fax, to the input supplied. Hence, a fax tool may take as input jsmith and compose the name user/jsmith/service/fax for the default fax of the user jsmith. Policies such as user/ and service/ are described in Part 2, “FNS Policies.”
Similarly, to access a person’s calendar, you just need to type the person’s user name. The application takes the input, jsmith, and uses it to construct the composite name of the object, in this case, user/jsmith/service/calendar, as shown in Figure 2-5.

![End-user input](End-user input)

![Application](Application)

![Federation of naming systems](Federation of naming systems)

![XFN API](XFN API)

**File System View**

Users and applications also experience federated naming through the file system. The initial context is located under /xfn in the root directory. For example, user jsmith’s to_do file has the XFN name, user/jsmith/fs/to_do. To read this file, you could type:

```
% cat /xfn/user/jsmith/fs/to_do
```

Applications access the files under /xfn just as they do any other files. Applications do not need to be modified in any way, nor do they need to use the XFN API.
Application View

The way that client applications interact with XFN to access different naming systems is illustrated in a series of figures. Figure 2-6 shows an application that uses the XFN API and library.

*Figure 2-6  Client Application Interaction With XFN*
Figure 2-7 shows the details beneath the API. A name service that is federated is accessed through the XFN client library and a context shared object module. This module translates the XFN calls into name service–specific calls.
X.500, DNS, and NIS+ are the name services that have been federated in the example shown in Figure 2-8.

As resolution of a composite name proceeds, it may cause these different modules to be linked in, depending on the types of contexts referenced in the name.

Figure 2-8  XFN Implementation Examples
API Usage Model

Many clients of the XFN interface are only interested in lookups. Their usage of the interface amounts to:

- Obtaining the initial context
- Looking up one or more names relative to the initial context

Once the client obtains a desired reference from the lookup operation, it constructs a client-side representation of the object from the reference. This need not be code within the application layer but may be code inside the service layer. For example, RPC services can provide clients with a means of constructing client-side handles from a composite name for the service or from a reference containing an RPC address for the service. After getting this handle, the client performs all further operations on the object or service by supplying the handle.

This is the basic model of how the XFN interface is expected to be used. The enterprise policies presented in Chapter 4 further encourage a bind/lookup model for how services and clients may rendezvous through the use of the name service.
Part 2 — FNS Policies

These three chapters examine the FNS naming policies that are integral to using FNS.

<table>
<thead>
<tr>
<th>Policy</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction to FNS Policies</td>
<td>25</td>
</tr>
<tr>
<td>Policies for the Enterprise Namespace</td>
<td>37</td>
</tr>
<tr>
<td>Policies for the Global Namespace</td>
<td>61</td>
</tr>
</tbody>
</table>
This chapter introduces FNS policies.

<table>
<thead>
<tr>
<th>Policy Overview</th>
<th>page 26</th>
</tr>
</thead>
<tbody>
<tr>
<td>Examples of Composite Names</td>
<td>page 29</td>
</tr>
<tr>
<td>How FNS Policies Relate to NIS+</td>
<td>page 30</td>
</tr>
<tr>
<td>Target Client Applications of FNS Policies</td>
<td>page 32</td>
</tr>
</tbody>
</table>

XFN defines policies for naming objects in the federated namespace. The goals of these policies are:

- To allow easy and uniform composition of names
- To promote coherence in naming across applications and services
- To provide a simple, yet sufficiently rich, set of policies so that applications need not invent and implement ad hoc policies for specific environments
- To enhance an application’s portability
- To promote cross-platform interoperability in heterogeneous computing environments

FNS policies contain all the XFN policies as well as extensions for the Solaris environment.
Policy Overview

Computing environments now offer worldwide scope and a large range of services. Users expect to have access to services at every level of the computing environment. Figure 3-1 on page 27 shows that FNS policies provide a common framework for the three levels of services: global, enterprise, and application.

What FNS Policies Specify

FNS provides applications with a set of policies on how name services are arranged and used at the enterprise level:

- Name services for enterprise objects: organizations, hosts, users, sites, and services. (these name services support contexts that allow other objects to be named relative to these objects)
- The relationships among the organization, host, user, site, and service name services, and the names used to refer to these name services
- The syntax of names in these name services
- How to federate the enterprise namespace so that it is accessible in the global namespace
- Names and bindings present in the initial context of every process

What FNS Policies Do Not Specify

The FNS policies do not specify the specific names used within name services. In addition, naming within the application is left to individual applications or groups of related applications.
Figure 3-1  Different Levels of Name Services
What FNS Enterprise Policies Arrange

The FNS enterprise policies deal with the arrangement of objects within the enterprise namespace. This section introduces the types of objects named within an enterprise and how they are arranged, as shown in Figure 3-2. These entities are described in greater detail in Chapter 4, “Policies for the Enterprise Namespace.” The policies are summarized in Table 4-2 on page 43.

- **Organization** — Entities such as departments, centers, and divisions. Sites, hosts, users, and services can be named relative to an organization. The XFN term for organization is organizational unit.
- **Site** — Physical locations, such as buildings, machines in buildings, and conference rooms within buildings. Sites may have files and services associated with them.
- **Host** — Computers. Hosts may have files and services associated with them.
- **User** — Human users. Users may have files and services associated with them.
- **Service** — Services such as printers, faxes, mail, and electronic calendars.
- **File** — Files within a file system.

![Figure 3-2  What FNS Policies Arrange](image-url)
Examples of Composite Names

This section shows examples of names that follow FNS policies. The specific choices of organization names, site names, user names, host names, file names, and service names (such as “calendar” and “printer”) are illustrative only; these names are not specified by FNS policy.

**Composing Names Relative to Organizations**

The naming systems to be found under an organization are: user, host, service, fs, and site.

- `org/csl.parc/site/videoconference.northwing` names a conference room `videoconference` located in the north wing of the site associated with the organization `csl.parc`.

- `org/csl.parc/user/mjones` names a user `mjones` in the organization `csl.parc`.

- `org/csl.parc/host/inmail` names a machine `inmail` belonging to the organization `csl.parc`.

- `org/csl.parc/fs/pub/blue-and-whites/CSL92-124` names a file `pub/blue-and-whites/CSL92-124` belonging to the organization `csl.parc`.

- `org/csl.parc/service/calendar` names the calendar service for the organization `csl.parc`. This might manage the meeting schedules for the organization.

**Composing Names Relative to Users**

The naming systems associated with users are service and fs.

- `user/jsmith/service/calendar` names the calendar service of the user `jsmith`.

- `user/jsmith/fs/bin/games/riddles` names the file `bin/games/riddles` under the home directory of the user `jsmith`.
Composing Names Relative to Hosts

The naming systems associated with hosts are service and fs.

host/mailhop/service/mailbox
  names the mailbox service associated with the machine mailhop.

host/mailhop/fs/pub/saf/archives.91
  names the directory pub/saf/archives.91 found under the root directory of the file system exported by the machine mailhop.

Composing Names Relative to Sites

The naming systems associated with sites are service and fs.

site/B5.MountainView/service/printer/speedy
  names a printer speedy in the B5.MountainView site.

site/B5.MountainView/fs/usr/dist
  names a file directory usr/dist available in the B5.MountainView site.

How FNS Policies Relate to NIS+

If you are not familiar with NIS+ and its terminology, refer to NIS+ and DNS Setup and Configuration Guide. You will find it helpful to be familiar with the structure of a typical NIS+ environment.

NIS+ Domains and FNS Organizational Units

FNS names organization, user, and host objects within the Solaris enterprise naming system, NIS+. An NIS+ domain is comprised of logical collections of users and machines and information about them, arranged to reflect some form of hierarchical organizational structure within an enterprise.

FNS is implemented on NIS+ by mapping NIS+ domains to FNS organizations. An organizational unit name corresponds to a NIS+ domain name and is identified using the fully qualified form of its NIS+ domain name, or its NIS+ domain name relative to the NIS+ root. The top of the FNS organizational namespace is mapped to the NIS+ root domain and is accessed using the name org/ from the initial context.
In NIS+, users and hosts have a notion of a home domain. It is the primary NIS+ domain that maintains information associated with them. A user or host’s home domain can be determined directly using its NIS+ principal name. An NIS+ principal name is composed of the atomic user (login) name or the atomic host name and the name of the NIS+ home domain. For example, user jsmith with home domain wiz.com. has an NIS+ principal name jsmith.wiz.com.

A user’s NIS+ home domain corresponds to the user’s FNS organizational unit. Similarly, a host’s home domain corresponds to its FNS organizational unit.

**Trailing Dot in Organization Names**

The trailing dot in an organization name indicates that the name is a fully qualified NIS+ domain name. Without the trailing dot, the organization name is an NIS+ domain name to be resolved relative to the NIS+ root domain.

For example, if the NIS+ root domain is wiz.com., with subdomains eng.wiz.com. and sales.wiz.com., the following pairs of names refer to the same organization:

- org/ org/wiz.com.
- org/eng org/eng.wiz.com.
- org/sales org/sales.wiz.com.

The name org/eng. (with trailing dot) would not be found, because there is no NIS+ domain with the eng. name.

**NIS+ Users and FNS Users**

Users in the NIS+ namespace are found in the passwd.org_dir table of a domain. Users in an FNS organization correspond to the users in the passwd.org_dir table of the corresponding NIS+ domain. FNS provides a context for each user in the passwd table.
NIS+ Hosts and FNS Hosts

Hosts in the NIS+ namespace are found in the hosts.org_dir table of a domain. Hosts in an FNS organization correspond to the hosts in the hosts.org_dir table of the corresponding NIS+ domain. FNS provides a context for each host in the hosts table.

Target Client Applications of FNS Policies

One goal of the FNS policies is to address coherence across the most commonly used tools, including the file system, the DeskSet™ tools, such as Calendar Manager, Print Tool, File Manager, and Mail Tool, and services that support these tools, such as RPC, email, and print subsystems.

---

**Note** – Some of these examples are not currently implemented in the Solaris environment. They are listed here as a way of illustrating how FNS may be used.

- **Calendars** — Instead of using names of the form `username@hostname` to access someone’s calendar, you would simply supply a user’s name, in most cases. You should also be able to use composite names to name calendars. For example, names of the following form would be accepted by calendar manager:

  jsmith
  user/mjones
  site/clarke.b5.mtv (calendar for Clarke conference room)

- **Printing** — Instead of naming a specific printer by its name, you should be able to name a printer relative to a user, site, or organization. For example:

  jsmith (jsmith’s default printer)
  org/dct (an organization’s default printer)
  site/clarke.b5.mtv (printer in the Clarke conference room)

- **File access** — You should be able to use composite names to name file systems and files. The automounter should use FNS to make resolution of composite names possible. For example, you should be able to use a file name like `/xfn/user/mbrown/fs/.cshrc` to reference the `cshrc` file for user mbrown.
• **RPC** — Instead of addressing services by their host name, program, and version numbers, you should be able to name the service using a composite name. For example, you should be able to name an RPC service relative to a user or an organization.

• **Mail** — Similarly, composite names can be used to name mail destinations. You should be able to use names such as the following:

```
jsmith
user/jsmith
org/dct (an organization’s mailing list)
site/clarke.b5.mtv (mailbox of the conference room coordinator)
```

• **Other desktop applications** — You should be able to pass composite names to other desktop applications such as spreadsheets, document preparation tools, fax tools, and so on. Some of these applications would attach their own namespace to the service namespace, thus becoming part of the FNS federation.

### Example Application: Calendar Service

This is a description of how one application, a calendar service, could be modified to use FNS policies. This example illustrates how FNS composite names might be presented to and accepted from users.

The DeskSet’s calendar service is typical of client-server applications. Calendar servers runs on some set of machines and maintain the users’ calendars. Calendar Manager (cm) runs on the desktop and contacts the appropriate server to obtain the calendars of interest.

The calendar service could benefit from FNS using a simple registry/lookup model as follows:

• ** Binding** — Upon startup, the server registers the calendars that it manages by binding a reference containing its own ONC+ RPC address (host, program, version) to the composite name for each calendar it manages, such as `user/jsmith/service/calendar`.

• **Lookup** — When using cm, the user specifies another user’s calendar simply by entering the user’s name (for example, jsmith) or selecting it from a list of names previously entered. Given the user name `jsmith`, `cm` composes
the composite name user/jsmith/service/calendar and uses this to look up the RPC address that it needs to communicate with the server that manages that calendar.

In the previous example, we used the name “calendar” to denote a calendar binding. The developers of the calendar service should register the name “calendar” with the FNS administrator, much as RPC programs are registered with the RPC administrator. Refer to “Service Name and Reference Registration” on page 40.

Note – The name “calendar” used here is an example. FNS policy does not specify the names of specific services.

The calendar service could take further advantage of FNS policy by allowing calendars to be associated with sites, organizations, and hosts, while still naming them in a uniform way. For example, by allowing calendars to be associated with a conference room (a site), the service can be used to “multibrowse” the conference room’s calendar as well as a set of user calendars to find an available time for a meeting in that room. Similarly, calendars can be associated with organizations for group meetings and hosts for keeping maintenance schedules.

_cm_ could simplify what the user needs to specify by following some simple steps.

1. _cm_ uses a tool for accepting composite names from the user and constructing the name of the object whose calendar is of interest. The object would be the name of a user, a site, a host, or an organization. For example, the user might enter the name _mjones_ and the calendar manager would generate the composite name _user/mjones_. This tool could be shared amongst a group of DeskSet applications.

2. _cm_ uses the XFN interface to compose this name with the suffix /service/calendar to obtain the name of the calendar.
3. This calendar name is then resolved relative to the process’s initial context. Continuing with the example, this would result in the resolution of the name user/mjones/service/calendar. Similarly, if the user enters the name of a site, clarke.B5.mtv, cm would generate the name site/clarke.B5.mtv/service/calendar for resolution.

Other services such as printing and mail could take advantage of the FNS policies in a similar way.
Policies for the Enterprise
Namespace

FNS policies specify the types and arrangement of namespaces within an enterprise and how such namespaces can be used by applications. This chapter describes these policies.

<table>
<thead>
<tr>
<th>Namespaces in the Enterprise</th>
<th>page 37</th>
</tr>
</thead>
<tbody>
<tr>
<td>Namespace Identifiers</td>
<td>page 41</td>
</tr>
<tr>
<td>Structure of the Enterprise Namespace</td>
<td>page 42</td>
</tr>
<tr>
<td>Initial Context Bindings for Naming Within the Enterprise</td>
<td>page 54</td>
</tr>
</tbody>
</table>

FNS policies described in this chapter include some extensions to XFN policy. These are explicitly defined with notes.

Namespaces in the Enterprise

This section introduces the types of namespaces within an enterprise:

- Organizational units
- Sites (an extension to the XFN policy)
- Hosts
- Users
- Services
- Files
- Printers (an extension to the XFN policy)
Some of these namespaces, such as users and hosts, can appear more than once in a federated namespace.

Organizational Unit Namespace

The organizational unit namespace provides a hierarchical namespace for naming subunits of an enterprise. Each organizational unit name is bound to an organizational unit context that represents the organizational unit.

In the Solaris environment, organizational units correspond to NIS+ domains and are named using dot-separated right-to-left compound names, where each atomic part names an organizational unit within a larger unit. For example, the name dct.eng names an organizational unit dct within a larger unit named eng.

Organizational unit names can be either fully qualified NIS+ domain names or relatively named NIS+ domain names. Fully qualified names have a terminal dot; relative names do not. If a terminal dot is present in the organization name, the name is treated as a fully qualified NIS+ domain name. If there is no terminal dot, the organization name is resolved relative to the top of the organizational hierarchy. For example, if the top organization corresponds to the NIS+ root domain wiz.com.(the trailing dot is significant) and if dct.eng is one of its suborganizations, the organization names dct.eng and dct.eng.wiz.com. (the trailing dot is significant) are equivalent.

Site Namespace

The site namespace provides a geographic namespace for naming objects that are naturally identified with their physical locations. These objects may be, for example, buildings on a campus, machines and printers in a building, conference rooms in a building and their schedules, and users in a particular quadrant of a building.

In the Solaris environment, sites are named using compound names, where each atomic part names a site within a larger site. The syntax of site names is dot-separated right-to-left, with components arranged from the most general to the most specific location description. For example, clarke.b5.mtv names the Clarke conference room in building 5 on the Mountain View campus of some enterprise.
User Namespace

The user namespace provides a namespace for naming human users in a computing environment.

Users are named in *username contexts*. The *username* context has a single-level namespace and contains bindings of user names to *user contexts*. A user context allows you to name objects relative to a user, such as files, services, or resources associated with the user.

In the Solaris environment, user names correspond to Solaris login user names.

Host Namespace

The host namespace provides a namespace for naming computers.

Hosts are named in *hostname contexts*. The *hostname* context has a flat namespace and contains bindings of host names to *host contexts*. A host context allows you to name objects relative to a host, such as files and printers found at that host.

In the Solaris environment, host names correspond to Solaris host names. Alias names for a single machine share the same context.

Service Namespace

The service namespace provides a namespace for services used by or associated with objects within an enterprise. Examples of such services are electronic calendars, faxes, mail, and printing.

In the Solaris environment, the service namespace is hierarchical. Service names are slash-separated (/) left-to-right compound names. An application that uses the service namespace can make use of this hierarchical property to reserve a subtree for that application. For example, the printer service reserves the subtree *printer* in the service namespace.

FNS does not specify how service names or reference types are chosen. These are determined by service providers that share the service namespace. For example, the calendar service uses the name *calendar* in the service context to name the calendar service and what is bound to the name *calendar* is determined by the calendar service.
Service Name and Reference Registration

SunSoft, Inc., maintains a registry of the names bound in the first level of the service namespace. To register a name, send an email request to fns-register@sun.com, or write to:

FNS Registration
Sun Microsystems, Inc.
2550 Garcia Avenue
Mountain View, CA 94043

Please include a brief description of the intended use of the name and a description of the format of the reference that can be bound to that name. See “References and Addresses” on page 153 for details on reference formats.

Printer Namespace

The printer namespace provides a namespace for naming printers. The printer namespace and context is described in more detail in Chapter 9, “Administering the Printer Namespace.”

File Namespace

A file naming system (or file system) provides a namespace for naming files. The file context and namespace is described in more detail in Chapter 8, “Administering the File System Namespace.”
Namespace Identifiers

The organizational unit namespace, site namespace (an extension to XFN policies), user namespace, host namespace, service namespace, and file namespace are referred to by the atomic names _orgunit, _site, _host, _user, _service, and _fs, respectively, in the federated enterprise namespace. These atomic names are called namespace identifiers.

Table 4-1  Namespace Identifiers in the Enterprise

<table>
<thead>
<tr>
<th>Namespace Identifier</th>
<th>Resolves to</th>
</tr>
</thead>
<tbody>
<tr>
<td>orgunit _orgunit</td>
<td>Context for naming organizational units</td>
</tr>
<tr>
<td>site _site</td>
<td>Context for naming sites</td>
</tr>
<tr>
<td>host _host</td>
<td>Context for naming hosts</td>
</tr>
<tr>
<td>user _user</td>
<td>Context for naming users</td>
</tr>
<tr>
<td>fs _fs</td>
<td>Context for naming files</td>
</tr>
<tr>
<td>service _service</td>
<td>Context for naming services</td>
</tr>
<tr>
<td>printer</td>
<td>Context for naming printers</td>
</tr>
</tbody>
</table>

In addition, FNS also supports the use of these names without the leading underscore (“_”) character (orgunit, site, host, user, service, and fs, respectively) as namespace identifiers for these namespaces. These names without the underscore are extensions to the XFN policies. The site and printer contexts are extensions to the XFN policies.
Note – In XFN terminology, the names with the leading underscore are the canonical namespace identifiers. The names without the underscore are the Solaris customized namespace identifiers. Solaris customized namespace identifiers, with the addition of printer, may not necessarily be recognized in non-Solaris environments. The canonical namespace identifiers are always recognized and, hence, portable to other environments.

The XFN component separator (/) is used to delimit namespace identifiers. For example, composing the namespace identifier orgunit with the organizational unit name accountspayable.finance gives the composite name, orgunit/accountspayable.finance.

Composing the namespace identifier user with the user name jsmith gives the composite name, user/jsmith.

FNS reserves the use of all atomic names in Table 4-1 as namespace identifiers in contexts in which namespace identifiers can appear, as defined by the arrangement of naming systems in “Structure of the Enterprise Namespace.” FNS does not otherwise restrict the use of these atomic names in other contexts. For example, the atomic name service is used as a namespace identifier relative to a user name, as in user/jsmith/service/calendar, to mean the root of user jsmith's service namespace. This does not preclude a system from using the name service as a user name, as in user/service, because FNS specifies that the context to which the name user/ is bound is for user names and not for namespace identifiers. Thus, in this case, service is unambiguously interpreted as a user name.

Structure of the Enterprise Namespace

FNS defines the structure of the enterprise namespace. The goal of this structure is to allow easy and uniform composition of names. This structure has two main rules: (1) objects with narrower scopes are named relative to objects with wider scopes, and (2) namespace identifiers are used to denote the
transition from one namespace to the next. Table 4-2 is a summary of FNS policy for arranging the enterprise namespace. Figure 4-1 on page 44 shows an example of a namespace layout that follows FNS policy.

Table 4-2 Policies for the Federated Enterprise Namespace

<table>
<thead>
<tr>
<th>Namespace Identifiers</th>
<th>Name Service Type</th>
<th>Subordinate Context</th>
<th>Parent Context</th>
<th>Namespace Organization</th>
<th>Syntax</th>
</tr>
</thead>
</table>
| orgunit _orgunit      | Organizational unit | Site, user, host, file system, service | Enterprise root | Hierarchical | NIS+ domain name  
Dot-separated, right-to-left |
| site _site            | Site               | Service, file system | Enterprise root, organizational unit | Hierarchical | Dot-separated, right-to-left |
| user _user            | User               | Service, file system | Enterprise root, organizational unit | Flat | Solaris login name |
| host _host            | Host               | Service, file system | Enterprise root, organizational unit | Flat | Solaris host name |
| service _service      | Service            | Application-specific | Enterprise root, organizational unit, site, user, host | Hierarchical | / separated, left-to-right |
| fs _fs                | File system        | None                | Enterprise root, organizational unit, site, user, host | Hierarchical | / separated, left-to-right |
| printer               | Printer            | None                | Service         | Hierarchical | / separated, left-to-right |
Figure 4-1  Example of an Enterprise Namespace

The namespace of an enterprise is structured around the hierarchical structure of organizational units of an enterprise. Names of sites, hosts, users, files, and services may be named relative to names of organizational units by composing the organizational unit name with the appropriate namespace identifier and object name.

In Figure 4-1, a user jsmith in the engineering organization of an enterprise is named using the name

```
orgunit/desktop.sw.eng/user/jsmith
```
Note the use of the namespace identifier user to denote the transition from the orgunit namespace to the user namespace. In a similar fashion (with the use of appropriate namespace identifiers), names of files and services may also be named relative to names of sites, users, or hosts. Names of sites may be named relative to organizational unit names.

The goal of easy and uniform composibility of names is met using this structure. For example, once you know the name for an organizational unit within an enterprise (for example, orgunit/sw), you can name a user relative to it by composing it with the user namespace identifier and the user’s login name to yield a name such as

```
orgunit/sw/user/joe
```

To name a file in this user’s file system, you can use a name like

```
orgunit/sw/user/joe/fs/notes
```

**Double Slashes in Organization Name**

orgunit// names the context of namespace identifiers associated with the root organizational unit, as in orgunit//service/printer. See “Significance of Double Slashes” on page 99 for more details on the significance of double slashes.

**Enterprise Root**

The root context of an enterprise, referred to as the enterprise root, is a context for naming objects found at the root of the enterprise namespace.

**Parent Context**

Enterprise roots are bound in the global namespace.

**Subordinate Contexts**

The following objects can be named relative to the enterprise root:

- Organizational units in that enterprise
- Sites in the top organizational unit of the enterprise (an extension to XFN policies)
• Users in the top organizational unit of the enterprise
• Hosts in the top organizational unit of the enterprise
• Services for the top organizational unit of the enterprise
• File service for the top organizational unit of the enterprise

These objects are named by composing with the namespace identifier of the target object’s namespace and the name of the target object.

For example, if \( .../wiz.com \) is the name of an enterprise, the root of the context for naming organizational units is

\( .../wiz.com/orgunit/ \)

and an organizational unit in that enterprise would be named using a name like

\( .../wiz.com/orgunit/finance \)

Sites in that enterprise would be named using a name starting with

\( .../wiz.com/site/ \)

Organizational Units

Parent Context

Organizational units may be named relative to the enterprise root.

Given an organization name, you can compose a name for its organizational unit context by using one of the namespace identifiers, \texttt{orgunit} or \texttt{_orgunit}. For example, if \( .../wiz.com \) is the name of an enterprise, the root of the context for naming organizational units is

\( .../wiz.com/orgunit/ \)

and an organizational unit in that enterprise would be named using a name like

\( .../wiz.com/orgunit/finance \)

---

1. The atomic name “...” (three dots) appears in the initial context to mean the global context. See Chapter 5, “Policies for the Global Namespace” for a description of the global context.
**Subordinate Contexts**

The following objects can be named relative to an organizational unit name:

- Sites for that organizational unit (an extension to the XFN policies)
- Hosts in that organizational unit
- Users in that organizational unit
- Services for that organization unit
- File service for that organizational unit

These objects are named by composing the organizational unit’s name with the namespace identifier of the target object’s namespace and the name of the target object. For example, the name

```
orgunit/sales/service/calendar
```

names the calendar service of the sales organizational unit. Similar, the name

```
orgunit/sales/user/jpeters
```

names a user *jpeters* in the sales organizational unit.

To name an organizational unit’s subunit, you use the organizational unit namespace’s syntax to compose the subunit’s name. For example, a subunit *accountspayable* of the organizational unit *finance* is named using the name

```
orgunit/accountspayable.finance
```

whereas user *jsmith* in the finance organizational unit is named using the name

```
orgunit/finance/user/jsmith
```

**Sites**

Sites are an extension to the XFN policies.

**Parent Contexts**

Sites may be named relative to

- The enterprise root
- An organizational unit
Sites named relative to the enterprise root are the same as sites named relative to the top organizational unit. Given an organization name, you can compose a name for its site context by using one of the namespace identifiers, site or _site. For example, if \(.../wiz.com\) is the name of an enterprise, the root of the context for naming sites is

\(.../wiz.com/site\)

which is equivalent to

\(.../wiz.com/orgunit//site\)

A site in that enterprise would be named using a name like

\(.../wiz.com/site/OceanSide\)

In another example, the name

\(orgunit/eng/site/\)

names the root of the site namespace for the organizational unit \(orgunit/eng\), while the name

\(orgunit/eng/site/b5.mtv\)

names a location within that site; in this example, building 5 in Mountain View.

**Subordinate Contexts**

The following objects can be named relative to a site name:

- Services at the site, such as the site schedule or calendar, printers, and faxes
- The file service available at the site

These objects are named by composing the site name with the namespace identifier of the target object’s namespace and the name of the target object.

For example, the name

\(site/Clark.b5.mtv/service/calendar\)

names the calendar service of the conference room \(Clark.b5.mtv\) and is obtained by composing the site name \(site/Clark.b5.mtv\) with the service name \(service/calendar\). To name a subunit of a site, you use the site namespace syntax. For example, the name
site/b5.mtv

names building 5, on the Mountain View campus of some enterprise while the name

site/mtv/service/calendar

names the calendar service for the Mountain View site.

Users

Parent Contexts

Users can be named relative to

• An organizational unit
• The enterprise root

Users named relative to the enterprise root are the same as users named relative to the top organizational unit. Given an organization name, you can compose a name for its username context by using one of the namespace identifiers, user or _user. Thus, if orgunit/dct.eng names an organization, then

    orgunit/dct.eng/user/

names the username context of dct.eng. The username context contains bindings of user names to user contexts.

Continuing with the above example, the name

    orgunit/dct.eng/user/jsmith

names a user jsmith in the dct.eng organizational unit.

Subordinate Contexts

The following objects can be named relative to a user name:

• Services associated with the user
• The user’s files
These objects are named by composing the user’s name with the namespace identifier of the target object’s namespace and the name of the target object. For example, the name

user/jjones/service/calendar

names the calendar for the user jjones.

Hosts

Parent Contexts

Hosts can be named relative to

- An organizational unit
- The enterprise root

Hosts named relative to the enterprise root are the same as hosts named relative to the top organizational unit. Given an organization name, you can compose a name for its hostname context by appending one of the namespace identifiers, host or _host. Thus if orgunit/dct.eng names an organization, then

org/dct.eng/host/

names the hostname context of dct.eng. The hostname context contains bindings of host names to host contexts. Continuing with the above example, the name

org/dct.eng/host/silver

names a machine silver in the dct.eng organizational unit.

Subordinate Contexts

The following objects can be named relative to a host name:

- Services associated with the host
- Files exported by the host
These objects are named by composing the host name with the namespace identifier of the target object’s namespace and the name of the target object. For example, the name

\texttt{host/silver/fs/release}

names the file directory \texttt{release} being exported by the machine \texttt{silver}.

Typically, resources should not be named relative to hosts but relative to more intuitive entities such as organizations, users, or sites. Dependence on host names forces the user to remember information that is often obscure and sometimes not very stable. For example, a user’s files may move from one host to another, because of hardware changes, file space usage, users who share the same file server, network reconfigurations, and so on. Yet if the files were named relative to the user, such changes would not affect how the files are named. There may be a few cases in which the use of host names is appropriate. For example, if a resource is available only on a particular machine and is tied to the existence of that machine, and there is no other logical way to name the resource relative to other entities, then it may make sense to name the resource relative to the host.

\textit{Services}

Service names should be registered with SunSoft, Inc., as directed in “Service Name and Reference Registration” on page 40.

\textit{Parent Contexts}

A service can be named relative to

- An organizational unit
- The enterprise root
- A user
- A host
- A site

Services named relative to the enterprise root are the same as services named relative to the top organizational unit.
The parent contexts allow services, such as Calendar Manager or printers, to be named relative to the parent object. A service context is named by using the namespace identifiers service or _service, relative to the organization, site, user, or host with which it is associated. For example, if orgunit/accountspayable.finance names an organizational unit, then

orgunit/accountspayable.finance/service/calendar
	names the calendar service of the organizational unit accountspayable.finance.

**Subordinate Contexts**

FNS does not restrict the types of bindings in the service namespace. Applications may create contexts of a type other than service contexts and bind them in the service namespace.

FNS supports the creation of generic contexts in the service context. A generic context is similar to a service context except that a generic context has an application-determined reference type. All other properties of a generic context are the same as a service context.

For example, a company, World Intrinsic Designs Corp (WIDC), reserves the name extcomm in the service namespace to refer to a generic context for adding bindings related to its external communications line of products. The context bound to extcomm is a generic context, with reference type WIDC_comm. The only difference between this context and a service context is that this context has a different reference type.

**Files**

**Parent Contexts**

A file namespace can be named relative to

- The enterprise root
- An organizational unit
- A user
- A host
- A site
Files named relative to the enterprise root are the same as files named relative to the top organizational unit. A file context is named by using the namespace identifiers \texttt{fs} or \texttt{_fs}, relative to the organization, site, user, or host with which it is associated. For example, if \texttt{orgunit/accountspayable.finance} names an organizational unit, then

\begin{verbatim}
  orgunit/accountspayable.finance/fs/
\end{verbatim}

names the file service of the organizational unit \texttt{accountspayable.finance}. In another example, if \texttt{user/jsmith} names a user \texttt{jsmith}, the name

\begin{verbatim}
  user/jsmith/fs/highlights95.mif
\end{verbatim}

names her file \texttt{highlights95.mif}. The file service of the user defaults to her home directory, as specified in the NIS+ \texttt{passwd} table.

\section*{Subordinate Contexts}

There can be no other type of context subordinate to a file system.

\section*{Printers}

The printer context is an extension of XFN policies.

\section*{Parent Contexts}

A printer namespace can be named in the service context.

A printer context is named by using the namespace identifier, \texttt{printer}, in the service context relative to

- An organizational unit
- A user
- A host
- A site

For example, if \texttt{orgunit/accountspayable.finance} names an organizational unit, then

\begin{verbatim}
  orgunit/accountspayable.finance/service/printer
\end{verbatim}

names the printer service of the organizational unit \texttt{accountspayable.finance}.
Subordinate Contexts

There can be no other type of context subordinate to a printer.

Initial Context Bindings for Naming Within the Enterprise

The XFN API provides a function, `fn_ctx_handle_from_initial()`, that allows the client to obtain an initial context object as a starting point for name resolution. The initial context contains bindings that allow the client application to (eventually) name any object in the enterprise namespace. Figure 4-2 on page 54 shows the same naming system as the one shown in Figure 4-1 on page 44, except that the initial context bindings are shaded and shown in italics.

![Diagram of initial context bindings](image)

*Figure 4-2  Example of Enterprise Bindings in the Initial Context*
The initial context has a flat namespace for namespace identifiers. The bindings of these namespace identifiers are summarized in Table 4-3 on page 55 and are described in more detail in subsequent sections. There are three categories of bindings:

- User-related bindings
- Host-related bindings
- “Shorthand” bindings

In Table 4-3, the user to which the bindings are related is denoted by U, and the host to which the bindings are related is denoted by H. Not all of these names need to appear in all initial contexts. For example, when a program is invoked by the superuser, none of the user-related bindings appears in the initial context. These bindings are described in more detail in the following sections.

### Table 4-3 Initial Context Bindings for Naming Within the Enterprise

<table>
<thead>
<tr>
<th>Namespace Identifier</th>
<th>Binding</th>
</tr>
</thead>
<tbody>
<tr>
<td>myself</td>
<td>U’s user context</td>
</tr>
<tr>
<td>_myself</td>
<td></td>
</tr>
<tr>
<td>thisuser</td>
<td></td>
</tr>
<tr>
<td>myens</td>
<td>The enterprise root of U</td>
</tr>
<tr>
<td>_myens</td>
<td></td>
</tr>
<tr>
<td>myorgunit</td>
<td>U’s distinguished organizational unit context; in the Solaris environment, the distinguished organizational unit is U’s NIS+ home domain</td>
</tr>
<tr>
<td>_myorgunit</td>
<td></td>
</tr>
<tr>
<td>thishost</td>
<td>H’s host context</td>
</tr>
<tr>
<td>_thishost</td>
<td></td>
</tr>
<tr>
<td>thisens</td>
<td>The enterprise root of H</td>
</tr>
<tr>
<td>_thisens</td>
<td></td>
</tr>
<tr>
<td>thisorgunit</td>
<td>H’s distinguished organizational unit context; in the Solaris environment, the distinguished organizational unit is H’s NIS+ home domain</td>
</tr>
<tr>
<td>_thisorgunit</td>
<td></td>
</tr>
<tr>
<td>user</td>
<td>The context in which users in the same organizational unit as H are named</td>
</tr>
<tr>
<td>_user</td>
<td></td>
</tr>
<tr>
<td>host</td>
<td>The context in which hosts in the same organizational unit as H are named</td>
</tr>
<tr>
<td>_host</td>
<td></td>
</tr>
<tr>
<td>org</td>
<td>The root context of the organizational unit namespace in H’s enterprise; in the Solaris environment, this corresponds to the NIS+ root domain</td>
</tr>
<tr>
<td>_orgunit</td>
<td></td>
</tr>
<tr>
<td>orgunit</td>
<td></td>
</tr>
<tr>
<td>_orgunit</td>
<td></td>
</tr>
</tbody>
</table>
In XFN terminology, the names with the leading underscore prefix are the canonical namespace identifiers. The names without the leading underscore, with the additions of org and thisuser, are Solaris customizations. Solaris customized namespace identifiers are not guaranteed to be recognized in other, non-Solaris environments. The canonical namespace identifiers are always recognized and, hence, portable to other environments.

**Note** — The current implementations of FNS does not support the addition or modification of names and bindings in the initial context.

### User-related Bindings

FNS assumes that there is a user associated with a process when fn_ctx_handle_from_initial() is invoked. This association is based on the effective user ID (euid) of the process. Although the association of user to process may change during the life of the process, the original context handle does not change.

In the following sections the user is denoted by “U.” FNS defines the following bindings in the initial context that are related to U.

**myself**

The namespace identifier myself (or either synonym _myself or thisuser) in the initial context resolves to the user context of U. For example, if U is jsmith, the name myself resolves in the initial context to jsmith’s user context, and the name

```
myself/fs/.cshrc
```

names the file .cshrc of jsmith.
myorgunit

FNS assumes that each user is affiliated with an organizational unit of an enterprise. A user may be affiliated with multiple organizational units, but there must be one that is distinguished, perhaps by its position in the organizational namespace or by the user’s role in the organization. In NIS+ terminology, this organizational unit corresponds to the user’s home domain.

The namespace identifier _myorgunit (or synonym _myorgunit) resolves in the initial context to the context of U’s distinguished organizational unit. For example, if U is the user jsmith, and jsmith’s home domain is accountspayable.finance, then myorgunit resolves in the initial context to the organizational unit context for accounts_payable.finance, and the name

   myorgunit/service/calendar

resolves to the calendar service of accounts_payable.finance.

myens

FNS assumes that there is an association of a user to an enterprise. This corresponds to the NIS+ hierarchy that holds myorgunit.

The namespace identifier _myens (and synonym _myens) resolves in the initial context to the enterprise root of the enterprise to which U belongs. For example, assume that U is jsmith and jsmith’s NIS+ home domain is accountspayable.finance, which in turn is in the NIS+ hierarchy with root domain name wiz.com. The name

   myens/orgunit/

resolves to the top organizational unit of wiz.com.

**Note** – Writers of set-user-ID programs need to be careful when using user-related composite names, such as myorgunit or myself/service, because these bindings depend on the effective user ID of a process. In cases where programs have a set-user-ID of root in order to access system resources on behalf of the caller, it will generally be desirable to call seteuid(getuid()) before calling fn_ctx_handle_from_initial().

---

**Policies for the Enterprise Namespace** 57
Host-related Bindings

A process is running on a particular host when `fn_ctx_handle_from_initial()` is invoked. In the following discussion this host is denoted by “H.” FNS defines the following bindings in the initial context that are related to H.

**thishost**

The namespace identifier `thishost` (or its synonym `_thishost`) is bound to the host context of H. For example, if the process is running on the machine `gofer`, `thishost` would be bound to the host context of `gofer`, and the name `thishost/service/calendar` would refer to the calendar service of `gofer`.

**thisorgunit**

FNS assumes that there is an association of a host to an organizational unit. A host may be associated with multiple organizational units, but there must be one that is distinguished. In NIS+ terminology, this organizational unit corresponds to the host’s home domain.

The namespace identifier `thisorgunit` (or its synonym `_thisorgunit`) resolves to H’s distinguished organizational unit. For example, if H is the machine `gofer`, and `gofer`’s NIS+ home domain is `desktop.engineering`, `thisorgunit` resolves to the organizational unit context for `desktop.engineering` and the name `thisorgunit/service/fax` refers to the fax service of the organizational unit `desktop.engineering`.

**thisens**

FNS assumes that there is an association of a host to an enterprise. This corresponds to the NIS+ hierarchy that holds `thisorgunit`.

The namespace identifier `thisens` (or its synonym `_thisens`) resolves to the enterprise root of H. For example, assume that H is the machine `gofer` and `gofer`’s NIS+ home domain is `desktop.engineering`, which in turn is in the NIS+ hierarchy with root domain name `wiz.com`. The name
thisens/site/

resolves to the root of the site namespace of wiz.com.

“Shorthand” Bindings

FNS defines the following “shorthand” bindings in the initial context to enable the use of shorter names to refer to objects in certain commonly referenced namespaces.

user

The namespace identifier user (or its synonym _user) is bound in the initial context to the username context in H’s organizational unit. This allows other users in the same organizational unit as H to be named from this context.

From the initial context, the names user and thisorgunit/user resolve to the same context. For example, if H is the machine gofer and gofer is in the desktop.engineering organizational unit, the name user/mjones names the user mjones in desktop.engineering.

host

The namespace identifier host (or its synonym _host) is bound in the initial context to the hostname context in H’s organizational unit. This allows other hosts in the same organizational unit as H to be named from this context.

From the initial context, the names host and thisorgunit/host resolve to the same context. For example, if H is the machine gofer and gofer is in the desktop.engineering organizational unit, the name host/bigbig names the machine bigbig in the organizational unit desktop.engineering.

org

The namespace identifier org (or its synonyms orgunit, _orgunit) is bound in the initial context to the root context of the organization naming system of the enterprise to which H belongs.

From the initial context, the names org and thisens/orgunit resolve to the same context. For example, if H is the machine gofer and gofer is in the enterprise wiz.com, the name

org/accountspayable.finance
names the organizational unit `accountspayable.finance` in `wiz.com`.

site

The namespace identifier `site` (or its synonym `_site`) is bound in the initial context to the root of the site naming system of the top organizational unit of the enterprise to which H belongs.

From the initial context, the names `site` and `thisens/site` resolve to the same context. For example, if H is the machine `gofer` and `gofer` is in the enterprise `wiz.com`, the name

```
site/clarke.b5.mtv
```

names a conference room, `clarke` in building 5 on the Mountain View campus of `wiz.com`.
This chapter describes the policies for naming objects that use global naming systems.

<table>
<thead>
<tr>
<th>Table 5-1</th>
<th>Policies for the Federated Global Namespace</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Namespace Identifier</strong></td>
<td><strong>Name Service Type</strong></td>
</tr>
<tr>
<td>. . .</td>
<td>Global</td>
</tr>
</tbody>
</table>

Global name services have worldwide scope. An enterprise “hooks up” to the federated global namespace by binding the root of the enterprise in the global namespace. This enables applications and users outside the enterprise to name objects within that enterprise. For example, a user within an enterprise can give out the global name of a file to a colleague in another enterprise to use.

DNS and X.500 contexts provide global-level name service for naming enterprises. FNS provides support for both DNS and X.500 contexts.
Initial Context Bindings for Global Naming

The atomic name “...” (three dots) appears in the initial context of every FNS client. The atomic name “...” is bound to a context from which global names can be resolved.

Global names can be either fully qualified Internet domain names or X.500 distinguished names.
- Internet domain names appear in the syntax specified by Internet RFC 1035.
- X.500 names appear in the syntax determined by the X/Open DCE Directory.

For example, .../wiz.com specifies a name to be resolved by DNS, whereas.../c=us/o=wiz specifies a name to be resolved by X.500.

The names “...” and “/...” are equivalent when resolved in the initial context. For example, the names /.../c=us/o=wiz and .../c=us/o=wiz resolve in the initial context to the same object.

Federating DNS

When a DNS name is encountered in the global namespace, it is resolved using the DNS name-resolution mechanism, the resolver library. The name typically resolves to an Internet host address or DNS domain records. Any fully qualified DNS names may be used in the global context. When the global context detects a DNS name, the name is passed to the DNS resolver for resolution. The result is converted into an XFN reference structure and returned to the caller.

The contents of DNS domains may be listed. However, the listing operations may be limited by practical considerations such as connectivity and security on the Internet. For example, listing the global root of the DNS domain is generally not supported by the root DNS servers. Most entities below the root, however, do support the list operation.

Table 5-2  Initial Context Bindings for Global Naming

<table>
<thead>
<tr>
<th>Atomic Name</th>
<th>Binding</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>Global context for resolving DNS or X.500 names</td>
</tr>
<tr>
<td>/...</td>
<td>Synonym for three dots</td>
</tr>
</tbody>
</table>

The atomic name “...” (three dots) appears in the initial context of every FNS client. The atomic name “...” is bound to a context from which global names can be resolved.

Global names can be either fully qualified Internet domain names or X.500 distinguished names.
- Internet domain names appear in the syntax specified by Internet RFC 1035.
- X.500 names appear in the syntax determined by the X/Open DCE Directory.

For example, .../wiz.com specifies a name to be resolved by DNS, whereas.../c=us/o=wiz specifies a name to be resolved by X.500.

The names “...” and “/...” are equivalent when resolved in the initial context. For example, the names /.../c=us/o=wiz and .../c=us/o=wiz resolve in the initial context to the same object.

Federating DNS

When a DNS name is encountered in the global namespace, it is resolved using the DNS name-resolution mechanism, the resolver library. The name typically resolves to an Internet host address or DNS domain records. Any fully qualified DNS names may be used in the global context. When the global context detects a DNS name, the name is passed to the DNS resolver for resolution. The result is converted into an XFN reference structure and returned to the caller.

The contents of DNS domains may be listed. However, the listing operations may be limited by practical considerations such as connectivity and security on the Internet. For example, listing the global root of the DNS domain is generally not supported by the root DNS servers. Most entities below the root, however, do support the list operation.
DNS hosts and domains are distinguished by the presence or absence of name service (NS) resource records associated with DNS resource names. If an NS record exists for a resource name, then that name is considered to be the name of the domain, and the returned reference is of type `inet_domain`. Otherwise, the returned reference is of type `inet_host`.

DNS may be used to federate other naming systems by functioning as a nonterminal naming system. For example, an enterprise naming system may be bound to `wiz.com` in DNS such that the FNS name `.../wiz.com/` refers to the root of that enterprise’s FNS namespace. The enterprise naming system is bound to a DNS domain by adding the appropriate text (TXT) records to the DNS map for that domain. When the FNS name for that domain includes a trailing slash (`/`), the TXT resource records are used to construct a reference to the enterprise naming system. Procedural information for federating an NIS+ domain under FNS is provided in “Federating NIS+ Under DNS” on page 112.

For general information about DNS, see `in.named(1M)` or the DNS chapters in *NIS+ and DNS Setup and Configuration Guide*.

**Federating X.500**

X.500 is a global directory service. It stores information and provides the capability to look up information by name as well as to browse and search for information. The information is held in a directory information base (DIB). Entries in the DIB are arranged in a tree structure. Each entry is a named object and comprises a defined set of attributes. Each attribute has a defined attribute type and one or more values.

An entry is unambiguously identified by a *distinguished name* that is the concatenation of selected attributes from each entry in the tree along a path leading from the root down to the named entry. For example, using the DIB shown in Figure 5-1 on page 64,

```
c=us/o=wiz
```

is a distinguished name of the `wiz` organization in the U.S. Users of the X.500 directory may interrogate and modify the entries and attributes in the DIB.
FNS federates X.500 by supplying the necessary support to permit namespaces to appear to be seamlessly attached below the global X.500 namespace.

For example, FNS facilitates “hooking” the enterprise naming system for the wiz organization below X.500. Starting from the initial context, an FNS name to identify the sales organizational unit of the wiz organization might be

\[ .../c=us/o=wiz/orgunit/sales \]

The name within the enterprise is simply concatenated onto the global X.500 name. (Note that FNS names use the name “...” in the initial context to indicate that a global name follows.)

Name resolution of FNS names takes place as follows. When an X.500 name is encountered in the global namespace, it is resolved using the X.500 name-resolution mechanism. One of three outcomes is possible:

- The full name resolves to an X.500 entry. This indicates that the entry is held in X.500. The requested FNS operation is then performed on that entry.
- A prefix of the full name resolves to an X.500 entry. This indicates that the remainder of the name belongs to a subordinate naming system.
The next naming system pointer (NNSP) to the subordinate naming system is examined to return the XFN reference. Name resolution then continues in the subordinate naming system. NNSP is discussed in “Composite Name Resolution” on page 163.

- An error is reported.

X.500 entries may be examined and modified using FNS operations (subject to access controls). However, it is not currently possible to list the subordinate entries under the root of the X.500 namespace by using FNS.
Part 3 — Administration

These chapters describe the administration tasks for setting up and maintaining FNS in different namespaces.

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Administering FNS on NIS+</td>
<td>69</td>
</tr>
<tr>
<td>Federating NIS+ With Global Naming Systems</td>
<td>111</td>
</tr>
<tr>
<td>Administering the File System Namespace</td>
<td>117</td>
</tr>
<tr>
<td>Administering the Printer Namespace</td>
<td>127</td>
</tr>
</tbody>
</table>
Administering FNS on NIS+

This chapter describes the setup and administration of the FNS implementation on top of the NIS+ environment. Use the following procedures for a standard setup (in which contexts are created automatically for you). If you wish to set up contexts individually, then specific procedures in “Creating FNS Contexts Individually” on page 73 will apply.

<table>
<thead>
<tr>
<th>Estimating Resource Requirements</th>
<th>page 70</th>
</tr>
</thead>
<tbody>
<tr>
<td>Setting Up NIS+ Service for FNS</td>
<td>page 70</td>
</tr>
<tr>
<td>Setting Up the FNS Namespace</td>
<td>page 71</td>
</tr>
<tr>
<td>Replicating FNS Service</td>
<td>page 72</td>
</tr>
</tbody>
</table>

The following sections contain information to aid you in administering the FNS namespace after it has been set up.

<table>
<thead>
<tr>
<th>Managing and Examining FNS Contexts</th>
<th>page 83</th>
</tr>
</thead>
<tbody>
<tr>
<td>Managing and Examining FNS Attributes</td>
<td>page 92</td>
</tr>
<tr>
<td>Maintaining Consistency Between NIS+ and FNS</td>
<td>page 94</td>
</tr>
<tr>
<td>Mapping FNS Contexts to NIS+ Objects</td>
<td>page 96</td>
</tr>
<tr>
<td>Browsing FNS Structures Using NIS+ Commands</td>
<td>page 97</td>
</tr>
<tr>
<td>Checking Access Control</td>
<td>page 98</td>
</tr>
<tr>
<td>Error Messages</td>
<td>page 100</td>
</tr>
<tr>
<td>Troubleshooting</td>
<td>page 104</td>
</tr>
</tbody>
</table>
Setting Up FNS

Setting up FNS involves preparing the NIS+ environment that FNS will use and then creating FNS contexts for organizations, users, hosts, services, and sites. Depending on the size of the organization, you should allow several hours for the FNS setup to be completed, not including the required hardware and software preparation or any NIS+ preparation.

Estimating Resource Requirements

Before proceeding with any installation procedure, you must first ensure that the machines on which NIS+ servers for supporting FNS will run have sufficient memory and disk storage.

For example, to support an FNS environment with 1200 users and hosts, you will need

- A minimum of 20 Mbytes of disk space beyond the space needed for NIS+
- An additional 40 Mbytes of swap space

Setting Up NIS+ Service for FNS

It is recommended, though not required, that NIS+ objects used by FNS and standard NIS+ domain information be supported on separate sets of servers. This avoids placing additional loads on the standard NIS+ service. It also allows you to keep the administration of FNS’s use of NIS+ and the standard NIS+ service separate.

All NIS+ objects used by FNS are kept under the ctx_dir directory of an NIS+ domain, at the same level as the domain’s org_dir directory. The NIS+ domain must be already set up before setting up FNS. That is, NIS+ domain tables, such as hosts and passwd, must already exist and be populated.

Before setting up the FNS namespace, do the following:

1. Set the NIS_GROUP environment variable to the name of the group that will be administering the FNS objects.

In fact, the fncreate command will not let you complete the FNS setup without setting the variable first. In this example, NIS_GROUP is set to fns_admins.wiz.com. When fncreate creates user and host contexts, they will be owned by those hosts and users, and not by the administrator.
who executed the command. Setting NIS_GROUP allows the administrators who are members of the group to subsequently modify these contexts even thought they do not own the objects.

```bash
# set NIS_GROUP=fns_admins.wiz.com;export NIS_GROUP
```

2. To set up separate servers, create the `ctx_dir` directory for the NIS+ domain. Assign a master server to service it with the NIS+ command `nismkdir`. The example shows how `nismkdir` creates the `ctx_dir` directory and assigns the machine `fns_mserver` as the master server for that directory. Include the trailing dot as shown.

```bash
# nismkdir -m fns_mserver ctx_dir.wiz.com.
```

3. Use the `nisls` command to verify that the `ctx_dir` directory has been created.

```bash
# nisls wiz.com.
ctx_dir
groups_dir
org_dir
```

### Setting Up the FNS Namespace

The FNS namespace is created by the `fncreate` command. This command creates the contexts for the specified organization and all its subcontexts, including contexts for users and hosts in the NIS+ domain corresponding to the organization.
1. For a standard setup, use the syntax of fncreate as shown.
This creates the organization context for the root NIS+ domain, wiz.com, contexts for all users found in the passwd.org_dir table, and contexts for all hosts found in the hosts.org_dir table in the wiz.com NIS+ domain.

```
# fncreate -t org org/wiz.com./
```

After setting up the FNS namespace, you should checkpoint the ctx_dir directory before performing other FNS operations.

2. Use nisping to checkpoint the ctx_dir directory:

```
```

For an organization with a few thousand users and hosts, the initial fncreate for an organization will typically take several hours; the subsequent checkpoint will also typically take several hours.

Repeating FNS Service

Additional replicas should be added to serve the ctx_dir directory after the FNS namespace has been set up on the master server. Replicas enhance availability and read performance of the servers.

1. Use the nismkdir -s command to add a replica fns_rserver for the ctx_dir directory and send the contents of the directory to the replica.

```
# nismkdir -s fns_rserver ctx_dir.wiz.com.
```

2. Checkpoint the ctx_dir directory periodically with the nisping command.
The recommended period is every few days. The period you choose depends on how frequently changes are made to the FNS namespace.

```
```

At this point, you are done with the initial FNS setup.
Creating FNS Contexts Individually

FNS contexts are created using the `fncreate(1M)` command. This section describes the `fncreate` command and its other options. Use this section to create FNS contexts individually rather than for the entire organization as described in “Setting Up the FNS Namespace” on page 71.

The `fncreate` command has the following syntax,

```
```

where `context_type` specifies one of the following: org, hostname, username, host, user, service, site, nsid, generic, or fs.

`fncreate` creates a context of the specified type and binds it to the given composite name. It also creates subcontexts for the context.
When creating contexts bound to namespace identifiers, the name without the underscore (for example, `user`) is used to create the context and the name with the underscore (for example, `_user`) is then bound to the reference of the newly created context. This is done regardless of whether the name with or without the underscore is specified in the command line.

For example, the command

```bash
# fncreate -t username org/sales/_user
```

creates a context for `org/sales/user` and adds a binding for `org/sales/_user` to the context of `org/sales/user`.

---

**Table 6-1 fncreate Command Options**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-t</code></td>
<td>Specifies the type of context to create.</td>
</tr>
<tr>
<td><code>-f</code></td>
<td>Creates a context for every host or user listed in <code>input_file</code>. This option can only be used with the <code>-t username</code> or <code>-t hostname</code> option and is useful for creating contexts for a subset of users and hosts found in the corresponding NIS+ passwd and hosts tables, respectively.</td>
</tr>
<tr>
<td><code>-o</code></td>
<td>Creates only the context specified. Without the <code>-o</code> option, subcontexts are created according to the FNS policies.</td>
</tr>
<tr>
<td><code>-r</code></td>
<td>Specifies the <code>reference_type</code> of the generic context being created. It can only be used with the <code>-t generic</code> option.</td>
</tr>
<tr>
<td><code>-s</code></td>
<td>Creates new contexts for composite names already in use. Otherwise, no new contexts are created for names already bound.</td>
</tr>
<tr>
<td><code>-D</code></td>
<td>Displays information about the NIS+ object associated with a context each time a context is created. This option is useful for debugging.</td>
</tr>
<tr>
<td><code>-v</code></td>
<td>Displays information about the creation as each context is created.</td>
</tr>
</tbody>
</table>

---

1. The `org` context is the exception where the contexts for hostname, username, and service are created but not populated.
Organization Context

Use the org type to create an organization context. The composite name must be that of an existing NIS+ domain. An NIS+ domain is an NIS+ directory with an org_dir subdirectory. Associated host and passwd tables for the domain must also exist.

Assume the root NIS+ domain is wiz.com. In the example

```bash
# fncreate -t org org/sales/
```

there must be an NIS+ domain named sales. When the new context is created, a ctx_dir directory, if it does not already exist, is created under the directory of the domain, sales.wiz.com. The previous example created an organization context for the composite name org/sales/ and, in addition, created hostname, username, and service subcontexts for it, which in turn, created host and user contexts, and service subcontexts for hosts and users.

Effectively, the following commands are run:

```bash
fncreate -t hostname org/sales/host/
fncreate -t username org/sales/user/
fncreate -t service org/sales/service/
```

When fncreate -o -t org is used, only the org context is created. The hostname, username, and service contexts are also created but not populated with host and user contexts.

The org context is owned by the administrator who executed the fncreate command, as are the hostname, username, and service subcontexts. The host and user contexts, however, and their subcontexts are owned by the hosts or users for which the contexts were created. In order for the administrator to subsequently manipulate host and user contexts, the NIS_GROUP environment variable must have been set accordingly at the time fncreate is executed. See “Setting Up NIS+ Service for FNS” on page 70 for instructions.
All Hosts Context

The hostname type creates a hostname context in which host contexts can be created and bound. Host contexts and their subcontexts are created for each host name found in the NIS+ hosts.org_dir table unless the -o option is used. When the -o option is used, only the hostname context is created.

For example, running the command

```
# fncreate -t hostname org/sales/host/
```

creates the hostname context and effectively runs the command

```
# fncreate -t hostname org/sales/host/hname/
```

for each host name, hname, found in the hosts.org_dir table. It also adds a binding for org/sales/_host/ that is bound to the reference of org/sales/host/.

The hostname context is owned by the administrator who executed the fncreate command. A host context and its subcontexts are owned by the host for which the contexts were created. That is, each host owns its own host context and subcontexts.

The -f option can be used to create contexts for a subset of the hosts found in the NIS+ table hosts.org_dir. It creates contexts for those hosts listed in the given input file.

Single Host Context

The host type creates the context and subcontexts for a host. The command automatically creates a service context for the host and a binding for fs unless the -o option is used. When the -o option is used, only the host context is created.

For example, the command

```
# fncreate -t host org/sales/host/capsule/
```
creates a context for the host named `capsule` and effectively runs the commands

```bash
fncreate -t service org/sales/host/capsule/service/
fncreate -t fs org/sales/host/capsule/fs/
```

The host context and its subcontexts are owned by the host. In the above example, the host `capsule`, with NIS+ principal name `capsule.sales.wiz.com`, owns the contexts `org/sales/host/capsule/`, `org/sales/host/capsule/service/`, and `org/sales/host/capsule/fs`.

The hostname context (`org/sales/host` in the above example) to which the host belongs must already exist. The host name supplied should already exist in the NIS+ `hosts.org_dir` table.

**Host Aliases**

Alias host names may exist in an NIS+ `hosts.org_dir` table. These appear in the table as a set of hosts with the same canonical name but different alias names.

In FNS, a single host with multiple alias names has a single host context. Alias names for that host in the `hostname` context are bound to the reference of that host context.

**All–Users Context**

The `username` type creates a username context in which user contexts can be created and bound. User contexts and their subcontexts are created for each user name found in the NIS+ `passwd.org_dir` table unless the `-o` option is used. When the `-o` option is used, only the `username` context is created.

For example, running the command

```bash
# fncreate -t username org/sales/user/
```
creates the *username* context and effectively runs the command

```
fncreate -t user org/sales/user/uname/
```

for each user, *uname*, that appears in the passwd.org_dir table. It also adds a binding for `org/sales/_user/` that is bound to the reference of `org/sales/user/`.

The *username* context is owned by the administrator who executed the `fncreate` command. A user context and its subcontexts are owned by the user for which the contexts were created. Each user owns his or her own user context and subcontexts.

The `-f` option can be used to create contexts for a subset of the users found in the NIS+ table passwd.org_dir. It creates contexts for those users listed in the given input file.

**Single User Context**

The *user* type creates the *user* context and subcontexts for a user. A service subcontext and a binding for *fs* are created under the *user* context unless the `-o` option is used. When the `-o` option is used, only the *user* context is created.

For example, the command

```
# fncreate -t user org/sales/user/jjones/
```

creates the *user* context for the user named *jjones* and effectively runs the commands

```
fncreate -t service org/sales/user/jjones/service/
fncreate -t fs org/sales/user/jjones/fs/
```

The *user* context and its subcontexts are owned by the user for whom the contexts were created. In the above example, the contexts created are owned by the user *jjones* with NIS+ principal name *jjones.sales.wiz.com*. 
The username context (org/sales/user in the above example) to which the user belongs must already exist. The user name supplied should already exist in the NIS+ passwd.org_dir table.

**Service Context**

The service type creates the service context in which service names can be bound. There is no restriction on what type of references may be bound in a service context. The policies depend on the applications that use the service context. For example, a group of desktop applications may bind references for a calendar, a telephone directory, a fax service, and a printer in a service context.

For example, the command

```bash
# fncreate -t service org/sales/service/
```

creates a service context for the organization sales. Because the terminal atomic name is a namespace identifier, fncreate also adds a binding for org/sales/_service/ that is bound to the reference of org/sales/service/.. After executing this command, names such as org/sales/service/calendar and org/sales/service/fax can then be bound in this service context.

The service context supports a hierarchical namespace, with slash-separated left-to-right names. The service namespace can be partitioned for different services. Continuing with the desktop applications example, a group of plotters may be named under the service context after the creation of the plotter context.

```bash
# fncreate -t service org/sales/service/plotter
```

Names such as org/sales/service/plotter/speedy and org/sales/service/plotter/production could then be bound under the service context.

**Note** – Because the terminal atomic name is not a namespace identifier, no additional binding is added (as was the case with service and _service).
The service context created is owned by the administrator who ran the fncreate command.

**Printer Context**

The printer context is created under the service context of the respective composite name. Refer to Chapter 9, "Administering the Printer Namespace," for more information.

**Generic Context**

The generic type creates a context for binding names used by applications.

A generic context is similar to a service context except it can have a different reference type. The \(-r\) option is used to specify the reference type for the generic context being created. If it is omitted, the reference type is inherited from its parent generic context or, if the parent context is not a generic context, the reference type used is a default generic reference type.

Like a service context, there is no restriction on what type of references may be bound in a generic context. The policies depend on the applications that use the generic context.

For example, the command

```
# fncreate -t generic -r WIDC_comm org/sales/service/extcomm
```

creates a generic context with the WIDC_comm reference type under the service context of the organization sales. Names such as `org/sales/service/extcomm/modem` can then be bound in this generic context.

The generic context supports a hierarchical namespace, with slash-separated left-to-right names, which allows an application to partition its namespace for different services. Continuing with the example above, a generic subcontext for modem can be created running the command

```
# fncreate -t generic org/sales/service/extcomm/modem
```
Names such as `org/sales/service/extcomm/modem/secure` and `org/sales/service/extcomm/modem/public` could then be bound under the `modem` context.

The generic context created is owned by the administrator who ran the `fncreate` command.

### Site Context

The `site` type creates contexts in which site names can be bound.

For example, the command

```bash
# fncreate -t site org/sales/site/
```

creates a site context. Because the terminal atomic name is a namespace identifier, `fncreate` also adds a binding for `org/sales/_site/` that is bound to the reference of `org/sales/site/`.

The `site` context supports a hierarchial namespace, with dot-separated right-to-left names, which allows sites to be partitioned by their geographical coverage relationships.

For example, the commands

```bash
# fncreate -t site org/sales/site/east
# fncreate -t site org/sales/site/maynard.east
```

create a site context `east` and a site subcontext `maynard.east` for it.

**Note** – Because these terminal atomic names are not namespace identifiers, no additional binding are added (as was the case with `site` and `_site`).

The `site` context created is owned by the administrator who ran the `fncreate` command.
File Context

The fs type creates a file system context (or file context) for a user or a host. For example, the command

```
# fncreate -t fs org/sales/user/jjones/fs/
```

creates the fs context for user jjones. Because the terminal atomic name is a namespace identifier, fncreate also adds a binding for org/sales/user/jjones/_fs/ that is bound to the reference of org/sales/user/jjones/fs/.

The fs context of a user is the user’s home directory as it is stored in the NIS+ passwd.org_dir table. The fs context of a host is the set of NFS file systems that the host exports (see NFS Administration Guide).

Use the fncreate_fs command to create file contexts for organizations and sites or to create file contexts other than the defaults for users and hosts. See Chapter 8, “Administering the File System Namespace,” for details.

The fs context created is owned by the administrator who ran the fncreate command.

Namespace Identifier Context

The nsid (namespace identifier) type creates a context in which namespace identifiers can be bound.

For example, the command

```
# fncreate -t nsid org/sales/site/maynard.east/
```

creates the nsid context for the site maynard.east and permits the creation of subcontexts such as service/. Continuing with this example, you could then execute the command

```
# fncreate -t service org/sales/site/maynard.east/service/
```

to create the service context for maynard.east.
The nsid context created is owned by the administrator who ran the fncreate command.

**Managing and Examining FNS Contexts**

A number of tools are provided for examining and managing FNS contexts. The commands and their syntax are shown as follows. For additional information, see the man page for the tool. Note that fnbind has two usages.

```plaintext
fnlookup [-v][-L] composite_name
fnlist [-l][-v] [composite_name]
fnbind [-s][-v][-L] name new_name
fnbind -r [-s][-v] new_name [-O | -U] ref_type {[-O | -U] address_type [-c|-x] address_contents}+
fnunbind composite_name
fnrename [-sv] context_name old_atomic_name new_atomic_name
fndestroy composite_name
```

**Displaying the Binding**

fnlookup displays the binding of the given composite name.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-v</td>
<td>Displays the binding in more detail</td>
</tr>
<tr>
<td>-L</td>
<td>Displays the reference to which the XFN link is bound</td>
</tr>
</tbody>
</table>
For example, the command

```
# fnlookup user/jjones/
Reference type: onc_fn_user
Address type: onc_fn_nisplus
  context type: user
```

shows the binding of the user jjones.

```
# fnlookup -v user/jjones/
Reference type: onc_fn_user
Address type: onc_fn_nisplus
  length: 52
  context type: user
  representation: normal
  version: 0
  internal name: fns_user_jjones.ctx_dir.sales.wiz.com.
```

Suppose user/James.Jones is linked to user/jjones. The first command in the following example shows what user/James.Jones is bound to (an XFN link). The second command follows the XFN link, user/jjones, and shows what user/jjones is bound to (the user context).

```
# fnlookup user/James.Jones
Reference type: fn_link_ref
Address type: fn_link_addr
  Link name: user/jjones

# fnlookup -L user/James.Jones
Reference type: onc_fn_user
Address type: onc_fn_nisplus
  context type: user
```
Listing the Context

fnlist lists the contents of the context identified by the given name.

Table 6-3  fnlist Command Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-v</td>
<td>Displays the binding in more detail</td>
</tr>
<tr>
<td>-l</td>
<td>Displays the bindings of the names bound in the named context</td>
</tr>
</tbody>
</table>

For example, the command

```bash
# fnlist user/
Listing 'user/':
jjones
mladd
jsmith
James.Jones
```

shows the bindings under the user context.
If no name is given, the command lists the contents of the initial context.

```
# fnlist
Listing '':
  _myorgunit
  ...
  _myself
  thishost
  myself
  _orgunit
  _host
  _thisens
  myens
  thisens
  org
  orgunit
  thisuser
  _thishost
  myorgunit
  _user
  thisorgunit
  host
  _thisorgunit
  _myens
  user
```
When the `-l` option is given, the bindings of the names bound in the named context are displayed.

```
# fnlist -l user/
Listing bindings 'user/':
name: mladd
  Reference type: onc_fn_user
  Address type: onc_fn_nisplus
  context type: user
name: jsmith
  Reference type: onc_fn_user
  Address type: onc_fn_nisplus
  context type: user
name: James.Jones
  Reference type: fn_link_ref
  Address type: fn_link_addr
  Link name: user/jjones
  name: jjones
name: jjones
  Reference type: onc_fn_user
  Address type: onc_fn_nisplus
  context type: user
```
When the `-v` option is given in conjunction with the `-l` option, the bindings are displayed in detail.

```bash
# fnlist -lv user/
Listing bindings 'user/':
name: mladd
Reference type: onc_fn_user
Address type: onc_fn_nisplus
  length: 52
  context type: user
  representation: normal
  version: 0
  internal name: fns_user_mladd.ctx_dir.sales.wiz.com.
name: jsmith
Reference type: onc_fn_user
Address type: onc_fn_nisplus
  length: 52
  context type: user
  representation: normal
  version: 0
  internal name: fns_user_jsmith.ctx_dir.sales.wiz.com.
name: James.Jones
Reference type: fn_link_ref
Address type: fn_link_addr
  length: 11
  data: 0x75 0x73 0x65 0x72 0x2f 0x6a 0x6f 0x65 0x6e 0x6f 0x66
  user/jjones
name: jjones
Reference type: onc_fn_user
Address type: onc_fn_nisplus
  length: 52
  context type: user
  representation: normal
  version: 0
  internal name: fns_user_jjones.ctx_dir.sales.wiz.com.
```
**Binding a Composite Name to a Reference**

`fnbind` allows you to bind a composite name to a reference. There are two uses of this command. The first usage allows the user to bind the reference of an existing name to a new name. The second usage allows the user to bind a reference constructed using arguments in the command line to a name.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-s</td>
<td>Supersedes any existing binding of the original composite name</td>
</tr>
<tr>
<td>-v</td>
<td>Prints out the reference used for the binding</td>
</tr>
<tr>
<td>-L</td>
<td>Creates an XFN link using <code>name</code> and binding it to <code>new_name</code></td>
</tr>
<tr>
<td>-c</td>
<td>Stores address contents without XDR encoding</td>
</tr>
<tr>
<td>-x</td>
<td>Interprets address contents as a hexadecimal input string and store it as is</td>
</tr>
<tr>
<td>-r</td>
<td>Creates a reference with a specified type and bind the reference to a name specified on the command line</td>
</tr>
<tr>
<td>-O</td>
<td>Interprets and stores type string as ASN.1 dot-separated integer list</td>
</tr>
<tr>
<td>-U</td>
<td>Interprets and stores type string as a DCE UUID</td>
</tr>
</tbody>
</table>

The first usage of `fnbind` is

```
fnbind [-s][-v][-L] name new_name
```

For example, the command

```
# fnbind myorgunit/service/printer user/mladd/service/printer
```

binds the name `user/mladd/service/printer` to the reference of `myorgunit/service/printer`.

If the given `new_name` is already bound, `fnbind -s` must be used or the operation will fail. In the above example, if `user/mladd/service/printer` is already bound, the `-s` option must be used to overwrite the existing binding with that of `myorgunit/service/printer`.

```
# fnbind -s myorgunit/service/printer user/mladd/service/printer
```
The –v option prints out the reference used for the binding.

```bash
# fnbind -v myorgunit/service/printer user/mladd/service/printer
Reference type: onc_printers
Address type: onc_fn_printer_nisplus
```

The following command constructs an XFN link out of `user/jjones` and binds it to the name `user/James.Jones`.

```bash
# fnbind -L user/jjones user/James.Jones
```

Similarly, the following creates a link from `user/mladd/service/printer` to `myorgunit/service/printer`.

```bash
# fnbind -sL myorgunit/service/printer user/mlad/service/printer
```

The second usage of `fnbind` constructs a reference using arguments in the command line and binds it to the given name.

```bash
fnbind -r [-s] [-v] new_name [-O | -U] ref_type {[-O | -U] | address_type [-c|-x] address_contents}+
```

For example

```bash
# fnbind -r thisorgunit/service/calendar onc_calendar onc_cal_str staff@exodus
```

binds the name `thisorgunit/service/calendar` to the reference with type `onc_calendar` and address type `onc_cal_str` and address contents of `staff@exodus`.

By default, the address contents supplied in the command line is XDR-encoded before being stored in the reference. If the `-c` option is given, the address contents are stored in the clear, not as an XDR-encoded string. If the `-x` option is given, the address contents supplied in the command line are interpreted as a hexadecimal string and stored (and not XDR-encoded).
By default, the reference and address types of the reference to be constructed uses the FN_ID_STRING identifier format. If the `-O` option is given, the identifier format is FN_ID_ISO_OID_STRING, an ASN.1 dot-separated integer list string. If the `-U` option is given, the identifier format is FN_ID_DCE_UUID, a DCE UUID² in string form. For example, the following command binds to the name thisorgunit/service/nx a reference with OIDs as reference and address types and a hexadecimal string as the address contents.

```
# fnbind -r thisorgunit/service/nx -O 1.2.99.6.2.1 -O 1.2.99.6.2.3 -x ef12eab67290
```

### Removing a Composite Name

`fnunbind` removes the given composite name from the namespace. Note that this does not remove the object associated with the name; it only unbinds the name from the object. For example, the command

```
# fnunbind user/jjones/service/printer/color
```

removes the binding associated with the name `user/jjones/service/printer/color`.

### Renaming an Existing Binding

`fnrename` renames an existing binding. The following example renames the binding of `clndr` to `calendar`, in the context named by `user/jjones/service/`.

```
# fnrename user/jjones/service/ clndr calendar
```

The `-s` option is used to overwrite any binding to `new_atomic_name`.

---

Destroying the Named Context

fndestroy removes the given composite name from the namespace and destroys the context named by the composite name.

For example, the command

```
# fndestroy user/jjones/
```

unbinds the name user/jjones/ from the namespace and destroys the context named by user/jjones/.

If the composite name identifies a context to be removed, the command fails if the context contains subcontexts.

Managing and Examining FNS Attributes

The fnattr command lets you update and examine attributes associated with FNS named objects. The four main options are for adding, deleting, listing, and modifying an attribute. In each of these cases, the identifier format is FN_ID_STRING, unless the option -O or -U is used.

Adding an Attribute

The -a option is for adding an attribute or adding a value to an attribute. You need to specify the composite name the attribute is associated with, the attribute identifier, and the values to add.

```
fnattr -a [-s] composite_name [-O | -U] identifier value1 [value2+]
```

The following example adds the attribute identifier model and the value hplaser to thisorgunit/service/printer.

```
# fnattr -a thisorgunit/service/printer model hplaser
```
The \(-s\) option means “add in supersede” mode. If an attribute with the specified identifier already exists, \(-s\) removes all of its values and replaces them with the values added. If this option is omitted, the resulting values for the specified attribute includes the existing values and the new values added.

```
# fnattr -as thisorgunit/service/printer model hplaser
```

The example above will first remove any existing values associated with model and add hplaser as the value.

**Deleting an Attribute**

To delete an attribute associated with an FNS named object, use the \(-d\) option. You can control what to delete:

- If an identifier is not specified, all the attributes associated with the named object are removed.
- If an identifier is specified, but without values, the entire attribute identified by `identifier` is removed.
- If individual values are specified, then only those values are removed from the attribute. Removal of the last value of an attribute is the same as removing the attribute itself.

```
fnattr -d composite_name [[-O | -U] identifier value1 [value2+]]
```

The following command deletes all the attributes associated with thisorgunit/service/printer.

```
# fnattr -d thisorgunit/service/printer
```

**Listing an Attribute**

The \(-l\) option is for listing attributes and their values. The command syntax is

```
fnattr -l composite_name [[-O | -U] identifier]
```
The following example lists the values of the `model` attribute of `thisorgunit/service/printer`.

```
# fnattr -l thisorgunit/service/printer model
laser
postscript
```

If an identifier is not specified, all the attributes associated with the named object are displayed.

**Modifying an Attribute**

The `-m` option lets you modify an attribute value. The command syntax is

```
fnattr -m composite_name [-O | -U] identifier old_value new_value
```

For example,

```
# fnattr -m thisorgunit/service/printer model postscript laser
```

replaces the value `postscript` with `laser`. Other attributes and values associated with `thisorgunit/service/printer` are not affected.

**Other Options**

The `-O` option assumes the format of the attribute identifier is an ASN.1 dot-separated integer string list (`FN_ID_ISO_OID_STRING`).

The `-U` option assumes the format of the attribute identifier is a DCE UUID string form (`FN_ID_DCE_UUID`).

**Maintaining Consistency Between NIS+ and FNS**

A key task of the system administrator is to maintain consistency between FNS and NIS+ by ensuring that the contents of FNS contexts and NIS+ tables correspond. This correspondence is initially accomplished by the `fncreate` command, which ensures that FNS contexts are correctly created and
populated and are consistent with NIS+ domain and table information. After the FNS contexts have been set up, the correspondence needs to be maintained as new users and hosts are added to and removed from the system.

The Solstice™ AdminTools™ product that adds user and host information to NIS+ also updates FNS. When updates to FNS or NIS+ are made independent of the Solstice AdminTools product, the resulting inconsistencies can be resolved by the use of the FNS tool, `fncheck`. `fncheck` checks for inconsistencies between user and host names in FNS, and user and host names in NIS+.

### Checking Naming Inconsistencies

The `fncheck` command checks for naming inconsistencies between the hostname and username contexts, and the NIS+ standard system tables `hosts.org_dir` and `passwd.org_dir`, respectively. `fncheck` lists host and user names that are in the FNS namespace but not in the NIS+ standard system tables. It also lists host or user names that are in the NIS+ standard system tables but not in the FNS namespace.

The command syntax is

```
fncheck[-r][-s][-u][-t hostname|username][domain_name]
```

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-t</td>
<td>Specifies the type of context to check</td>
</tr>
<tr>
<td>-s</td>
<td>Lists host or user names from the NIS+ standard system tables that are not in the FNS namespace</td>
</tr>
<tr>
<td>-r</td>
<td>Lists host or user names from the FNS namespace that do not have entries in the corresponding NIS+ standard system tables</td>
</tr>
<tr>
<td>-u</td>
<td>Updates the FNS namespace based on information in the relevant NIS+ standard system tables</td>
</tr>
</tbody>
</table>

The `-t` option is used to specify the contexts to check. For the `-t hostname` option, the hostname context associated with the organization is checked against the NIS+ `hosts.org_dir` table of the same organization. For the `-t username` option, the username context associated with the organization is
checked against the NIS+ passwd.org_dir table in the same organization. When the -t option is omitted, both the hostname and username contexts are checked.

When the -r option is used in conjunction with the -u option, items that appear only in the FNS context are removed from the FNS context. When the -s option is used in conjunction with the -u option, items that appear only in the NIS+ standard system tables are added to the FNS context. If neither -r or -s are specified, items are added and removed from the FNS context to make it consistent with the corresponding NIS+ table.

**Advanced FNS and NIS+ Issues**

This section provides detailed information on the relationship between NIS+ objects and FNS objects. This information is useful when you must change the access control of FNS objects.

**Mapping FNS Contexts to NIS+ Objects**

FNS contexts are stored as NIS+ objects. All contexts associated with an organization are stored under the ctx_dir directory of the associated NIS+ domain, which resides at the same level as the org_dir directory of the same domain.

Use the -v option for the fnlookup or fnlist command to see the detailed description of references. The internal name field displays the name of the corresponding NIS+ object.
Browsing FNS Structures Using NIS+ Commands

The NIS+ command, `nisls`, can be used to list the NIS+ objects used by FNS. For example, the following commands list the contents of the NIS+ domain directory and its `ctx_dir` subdirectory.

```
# nisls wiz.com.
wiz.com.:  
  eng  
  sales  
  org_dir  
  ctx_dir
```

```
# nisls ctx_dir.wiz.com.
ctx_dir_Wiz.COM.:  
  fns  
  fns_user  
  fns_host  
  fns_host_alto  
  fns_host_mladd  
  fns_host_elvira  
  fns_user_jjones  
  fns_user_jsmith  
  fns_user_aw
```

Use the `niscat` command to list the contents of the `fns_hosts` table.

```
# niscat fns_host.ctx_dir
  alto *BINARY* *BINARY*  
  mladd *BINARY* *BINARY*  
  elvira *BINARY* *BINARY*  
```
## Checking Access Control

Use `niscat -o` to see the access control of a context.

```
# niscat -o fns_host.ctx_dir
Object Name:fns_host
Owner: alto.wiz.com.
Access Rights:r-c-rmdrmdr-c-
Time to Live:53:0:56
Object Type:TABLE
Table Type:H
Number of Columns:3
Character Separator:
Search Path:
Columns:
[0] Name: atomicname
   Attributes:(SEARCHABLE, TEXTUAL DATA,CASE INSENSITIVE)
   Access Rights:r-c-rmdrmdr-c-
[1] Name: reference
   Attributes:(BINARY DATA)
   Access Rights:r-c-rmdrmdr-c-
[2] Name: flags
   Attributes:(BINARY DATA)
   Access Rights:r-c-rmdrmdr-c-
```
To see the access control of a particular binding, use the name of the binding entry in the parent context's binding table (that is, the name displayed in the internal name field in the output of `fnlookup -v` and `fnlist -v`):

```
# niscat -o "[atomicname=alto],fns_host.ctx_dir"
Object Name:fns_host
Owner: alto.wiz.com.
Access Rights:r-c-rmcdrmcd-r-c-
Time to Live:12:0:0
Object Type:ENTRY
  Entry data of type H
  [2] - [104 bytes] '0x00 ...'
  [3] - [1 byte] 0x01
```

To change the access control or ownership of a particular context, use the commands

- nischown
- nischmod
- nischgrp

and give as an argument either the binding entry or the bindings table, depending on the object the operation is to affect.

**Significance of Double Slashes**

In the name, `org//`, the double slashes identifies the context of namespace identifiers associated with the root organizational name, as in `org//service/printer`.

In contrast, `org/` points to the root of an organizational context. Each organizational context has suborganizations as well as a pointer to the context that contains namespace identifiers such as `service`, `user`, and `host`. `org/` names the root organizational context in which you can name suborganizations, as in `org/sales.finance`.
Significance of Trailing Slash

The trailing / names objects in the next naming system. You need it whenever you are going from one naming system to another. For example, the name, org/sales.finance names the context for naming suborganizations of the sales.finance organization, as in org/audit.sales.finance.

On the other hand, org/sales.finance/ names the context for naming namespace identifiers of the sales.finance organization, as in org/sales.finance/service/printer.

Error Messages

When an error occurs, FNS commands print out the remaining part of the name on which the operation failed. The part of the name that has not been printed has been processed successfully.

For example, a user attempted to create a context for org//service/trading/bb. The name org//service/ was resolved successfully, but trading was not found in the context named by org//service/. Thus, trading/bb is displayed as the part of the name that remains when the operation failed:

```
Error in creating 'org//service/trading/bb': Name Not Found: 'trading/bb'
```

In another example, a user attempted to destroy the context org//service/dictionary/english, but could not carry out the operation because the context named was not empty. The pair of single quotes (') indicates that FNS was able to resolve the complete name given, but could not complete the operation as requested:

```
Error in destroying 'org//service/dictionary/english': Context Not Empty: ''
```
**FNS Message Descriptions**

The FNS messages and their descriptions are as follows. These messages are encoded in the `FN_status_t` object as status codes. See either Table 10-3 on page 151 or the `FN_status_t(3)` man page for the corresponding status codes.

- **attribute no permission**
  
The caller did not have permission to perform the attempted attribute operation.

- **attribute value required**
  
The operation attempted to create an attribute without a value, and the specific naming system does not allow this.

- **authentication failure**
  
The operation could not be completed because the principal making the request cannot be authenticated with the name service involved. If the service is NIS+, check that you are identified as the correct principal (run the command `nisdefaults`) and that your machine has specified the correct source for public keys. Check that the `/etc/nsswitch.conf` file has the entry, `publickey: nisplus`.

- **bad reference**
  
FNS could not interpret the contents of the reference. This may result if the contents of the reference has been corrupted or when the reference identifies itself as an FNS reference, but FNS doesn’t know how to decode it.

- **communication failure**
  
FNS could not communicate with the name service to complete the operation.

- **configuration error**
  
An error resulted because of configuration problems. Examples: (1) the bindings table are removed out-of-band (outside of FNS), (2) and a host is in the NIS+ hosts table but does not have a corresponding FNS host context.
context not empty
   An attempt has been made to remove a context that still contains bindings.

continue operation using status values
   The operation should be continued using the remaining name and the resolved reference returned in the status.

error
   An error that cannot be classified as one of the other errors listed above occurred while processing the request. Check the status of the name services involved in the operation and see whether any of them are experiencing extraordinary problems.

illegal name
   The name supplied is not a legal name.

incompatible code sets
   The operation involved character strings from incompatible code sets, or the supplied code set is not supported by the implementation.

insufficient resources
   The name service used by FNS does not have sufficient resources to complete the request. Check memory and disk availability on the name servers involved.

invalid attribute identifier
   The attribute identifier is in a format not acceptable to the naming system, or its contents are not valid for the format specified for the identifier.

invalid attribute value
   The value supplied is not in the correct form for the given attribute.

invalid enumeration handle
   The enumeration handle supplied is invalid. The handle could have been from another enumeration, an update operation may have occurred during the enumeration, or there may have been some other reason.
invalid syntax attributes
The syntax attributes supplied are invalid or insufficient to fully specify the syntax.

link error
An error occurred while resolving an XFN link with the supplied name.

link loop limit reached
A nonterminating loop was detected due to XFN links encountered during composite name resolution, or the implementation-defined limit was exceeded on the number of XFN links allowed for a single operation.

malformed link
A malformed link reference was found during an fn_ctx_lookup_link() operation. The name supplied resolved to a reference that was not a link.

name in use
The name supplied is already bound in the context.

name not found
The name supplied was not found.

no permission
The operation failed because of access control problems.

no such attribute
The object did not have an attribute with the given identifier.

no supported address
No shared library could be found under the /usr/lib/fn directory for any of the address types found in the reference bound to the FNS name. Shared libraries for an address type are named according to this convention:
fn_ctx_address_type.so.

For example, a reference with address type onc_fn_nisplus would have a shared library in the path name:
/usr/lib/fn/fn_ctx_onc_fn_nisplus.so.
not a context
   The reference does not correspond to a valid context.

operation not supported
   The operation is not supported by the context. For example, trying to
   destroy an organization is not supported.

partial result returned
   The operation returned a partial result.

success
   Operation succeeded.

syntax not supported
   The syntax type is not supported.

too many attribute values
   The operation attempted to associate more values with an attribute than the
   naming system supports.

unavailable
   The name service upon which the operation depends is unavailable.

Troubleshooting

This section presents problem scenarios with a description of probable causes,
diagnoses, and solutions.

Cannot Obtain Initial Context

Symptom
   I get the message “Cannot obtain initial context.”

Possible Cause
   This is caused by an installation problem.
Diagnosis
Check that FNS has been installed properly by looking for the file, /usr/lib/fn/fn_ctx_initial.so.

Solution
Install the fn_ctx_initial.so library.

Nothing in Initial Context

Symptom
I run fnlist to look at what is in the initial context but see nothing.

Possible Cause
This is caused by an NIS+ configuration problem. The organization associated with the user and machine running the fn* commands do not have an associated ctx_dir directory.

Diagnosis
Use the nisls command to see whether there is a ctx_dir directory.

Solution
If there is no ctx_dir directory, run fncreate -t org/nis+_domain_name/ to create the ctx_dir directory.

“No Permission” Messages

Symptom
I get “no permission” messages.

Possible Cause
“No permission” messages mean that you do not have access to perform the command.
Diagnosis
Check permission using the appropriate NIS+ commands, described in “Advanced FNS and NIS+ Issues” on page 96. Use the nisdefaults command to determine your NIS+ principal name.

Another area to check is whether you are using the right name. For example, org// names the context of the root organization. Make sure you have permission to manipulate the root organization. Or maybe you meant to specify myorgunit/, instead.

Solution
If you do have permission, then the appropriate credentials probably have not been acquired.

This could be caused by the following:
- A keylogin has not been performed (defaults to NIS+ principal nobody).
- A keylogin was made to a source other than NIS+.
  - Check that the /etc/nsswitch.conf file has a publickey: nisplus entry.
  - This might manifest itself as an authentication error.

fnlist Does Not List Suborganizations

Symptom
I run fnlist with an organization name, expecting to see suborganizations, but instead see nothing.

Possible Cause
This is caused by an NIS+ configuration problem. Suborganizations must be NIS+ domains. By definition, an NIS+ domain must have a subdirectory named org_dir.

Diagnosis
Use the nisls command to see what subdirectories exist. Run nisls on each subdirectory to verify which subdirectories have an org_dir. The subdirectories with an org_dir are suborganizations.
Solution
Not applicable.

Cannot Create Host- or User-related Contexts

Symptom
When I run `fncreate -t` for the user, username, host, or hostname contexts, nothing happens.

Possible Cause
You have not set the NIS\_GROUP environment variable. When you create a user or host context it is owned by the host or user, and not by the administrator who set up the namespace. Therefore, `fncreate` requires that the NIS\_GROUP variable be set to enable the administrators who are part of that group to subsequently manipulate the contexts.

Diagnosis
Check the NIS\_GROUP environment variable.

Solution
The NIS\_GROUP environment variable should be set to the group name of the administrators who will administer the contexts.

Cannot Remove a Context I Created

Symptom
When I run `fndestroy` on the host or user context the context is not removed.

Possible Cause
You do not own the host or user context. When you create a user or host context it is owned by the host or user, and not by the administrator who set up the namespace.
Diagnosis
Check the NIS_GROUP environment variable.

Solution
The NIS_GROUP environment variable needs to be set to the group name of the administrator who will administer the contexts.

“Name in Use” With fnunbind

Symptom
I get “name in use” when trying to remove bindings. It works for certain names but not for others.

Possible Cause
You cannot unbind the name of a context. This restriction is in place to avoid leaving behind contexts that have no name (“orphaned contexts”).

Diagnosis
Run the fnlist command on the name to verify that it is a context.

Solution
If the name is a context, use the fndestroy command to destroy the context.

“Name in Use” With fnbind/fncreate -s

Symptom
I use the -s option with fnbind and fncreate, but for certain names I get “name in use.”

Possible Cause
fnbind -s and fncreate -s overwrite the existing binding if it already exists; but if the old binding is one that must be kept to avoid orphaned contexts, the operation fails with a “name in use” error because the binding could not be removed. This is done to avoid orphaned contexts.
Diagnosis
Run the `fnlist` command on the name to verify that it is a context.

Solution
Run the `fndestroy` command to remove the context before running `fnbind` or `fncreate` on the same name.

`fndestroy/fnunbind Does Not Return “Operation Failed”`

Symptom
When I do an `fndestroy` or `fnunbind` on certain names that I know do not exist, I receive no indication that the operation failed.

Possible Cause
The operation did not fail. The semantics of `fndestroy` and `fnunbind` are that if the terminal name is not bound, the operation returns success.

Diagnosis
Run the `fnlookup` command on the name. You should receive the message, “name not found.”

Solution
Not applicable.
Federating NIS+ With Global Naming Systems

FNS supports federation of enterprise naming systems implemented using NIS+ into the global naming systems, DNS and X.500. This chapter describes the procedures for federating NIS+ with DNS and X.500. In general, the procedures involve:

- Determining the NIS+ root reference for your NIS+ hierarchy
- Adding this information in the format required by the global naming system

Obtaining the NIS+ Root Reference

To federate NIS+ under DNS or X.500, information must be added to these respective naming systems to enable access to an NIS+ hierarchy from outside of the NIS+ hierarchy. This information comes from the NIS+ root reference, which consists of network address information describing how to reach the top of a particular NIS+ hierarchy.

The NIS+ root reference consists of a single address. The address has an address type of onc_fn_nisplus_root and contains a single, XDR-encoded string. The three items in the network address are separated by white spaces:

```
nis+_root_domain nis+_server [server_IP_address]
```
Table 7-1 is a description of the NIS+ root reference.

<table>
<thead>
<tr>
<th>Address Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nis+_root_domain</td>
<td>The fully qualified name of the NIS+ root domain (trailing dot required)</td>
</tr>
<tr>
<td>nis+_server</td>
<td>The host name of one of the servers serving nis+_root_domain</td>
</tr>
<tr>
<td>server_IP_address</td>
<td>The IP address of nis+_server. This is optional if the address of nis+_server is expected to be known. This means it should be available through one of the name services listed in the /etc/nsswitch.conf file. Refer to the nsswitch.conf(4) man page for information.</td>
</tr>
</tbody>
</table>

In the following example,

```
wiz.com. wiz-nis-master.wiz.com
```

the address indicates that name of the NIS+ root domain is wiz.com. (trailing dot is significant), and that it can be reached using the host wiz-nis-master.wiz.com. The IP address of the server is not given because it is expected to be available through other means.

In another example,

```
woz.com. wozwoz 133.33.33.33
```

indicates that the name of the NIS+ root domain is woz.com. (trailing dot is significant) and that it can be reached using the host wozwoz, with the IP address 133.33.33.33.

**Federating NIS+ Under DNS**

This section describes the steps required to add TXT (text) records for a subordinate enterprise naming system implemented with NIS+. To federate a subordinate naming system in DNS, you need to add reference information into DNS describing how to reach the subordinate naming system.
1. Obtain the NIS+ root reference for your NIS+ hierarchy, see “Obtaining the NIS+ Root Reference” on page 111.

2. Edit the DNS table (/etc/named.local is the default file name) and add a TXT record with the following format.

```
TXT "XFNNISPLUS nis+_root_domain nis+_server [server_IP_address]"
```

For more information about DNS tables, see *NIS+ and DNS Setup and Configuration Guide*.

The following are examples of two records that convey the same information.

```
TXT "XFNNISPLUS wiz.com. nis-master.wiz.com"
TXT XFNNISPLUS wiz.com. nis-master.wiz.com
```

The TXT record must be associated with a DNS domain that includes an NS (name server) record entry. The following is an example of a DNS table with reference information for NIS+ bound in it.

```
$ORIGIN Wiz.com
@ IN SOA foo bar.eng.Wiz.com
{ 100 ;; Serial 3600 ;; Refresh 3600 ;; Retry 3600 ;; Expire 3600 ;; Minimum }
NS nshost
TXT "XFNNISPLUS wiz.com. wiz-nis-master 133.33.33.33"
nshost IN A 133.33.33.34
```
3. After adding the TXT record into the DNS table, either restart the DNS server or send it a signal to reread the table.

```bash
# kill -HUP pid-of-in.named
```

For further information on how DNS TXT records are used for XFN references, see Appendix B.

**Federating NIS+ Under X.500**

In order to federate a subordinate naming system in X.500, reference information must be added into X.500 describing how to reach that subordinate naming system. This section describes the steps for adding XFN reference information to the X.500 entry that will be the parent of the subordinate naming system.

**Note** – An X.500 client is required in order to access X.500 using FNS. The X.500 client must export the XDS/XOM APIs from the `/opt/SUNWxds/lib/libxomxds.so` shared object. Consult “Getting started with the SunLink X.500 Client Toolkit” for details on SunSoft’s X.500 product.

1. **Obtain the NIS+ root reference for your NIS+ hierarchy.**
   See “Obtaining the NIS+ Root Reference” on page 111.

2. **Create an X.500 entry that supports XFN reference attributes.**
   For example, the following command creates a new X.500 entry called `c=us/o=wiz` with the object classes `top`, `organization`, and `XFN-supplement (1.2.840.113536.25)`. The `XFN-supplement` object class allows the `c=us/o=wiz` entry to store reference information for a subordinate naming system.

```bash
# fnattr -a .../c=us/o=wiz object-class top organization XFN-supplement
```

If the X.500 entry already existed and was not defined with the `XFN-supplement` object class, it must be removed and re-created with the additional object class. Otherwise, it will not be able to hold reference information about the subordinate naming system.
3. Add the reference information about the subordinate NIS+ system to the entry.
After creating the X.500 entry, you can then add information about the subordinate NIS+ system by binding the appropriate NIS+ root reference to the named entry:

```
# fnbind -r .../c=us/o=wiz/ onc_fn_enterprise onc_fn_nisplus_root "wiz.com. bigbig"
```

This example binds the reference for the NIS+ hierarchy with the root domain name `wiz.com`, served by the machine `bigbig`, to the next naming system pointer (NNSP) of the X.500 entry `c=us/o=wiz`, thus linking the X.500 namespace with the `wiz.com`. NIS+ namespace hierarchy.

The address format used is that of the NIS+ root reference described earlier. Note the use of the trailing slash in the name argument to `fnbind, .../c=us/o=wiz/`, to signify that the reference is being bound to the NNSP of the entry, rather than to the entry itself.

For further information on X.500 entries and XFN references, see Appendix C, “X.500 Attribute Syntax for XFN References.”
This chapter describes the file system namespace and the procedures for creating file contexts.

The FNS File System Namespace

Files may be named relative to users, hosts, organizations, and sites in FNS by appending the fs namespace identifier to the name of the object, and following this with the name of the file. For example, an engineering organization’s tools directory might be named org/engineering/fs/tools.

The initial context is located under /xfn in the file system’s root directory. Thus a user might access the tools directory by typing

```
% cd /xfn/org/engineering/fs/tools
```

Existing applications can access this directory just as they would any other directory. Applications do not need to be modified in any way or use the XFN API.
NFS File Servers

NFS is Sun’s distributed file system. The files associated with an object will generally reside on one or more remote NFS file servers. In the simplest case, the namespace identifier fs corresponds to the root of an exported NFS file system, as shown in Figure 8-1.

In contrast, an object’s file system may be composed of multiple—and possibly overlapping—remote mounts, woven together into a “virtual” directory structure managed by FNS.

Figure 8-1  NFS File System—Simple Case
Figure 8-2 illustrates how this capability might be used to piece together an organization's file system from three separate file servers. The project directory, along with its lib subdirectory, resides on one file server, while the src subdirectory resides on another. Users and applications need not be aware of the use of multiple servers; they see a single, seamless namespace.

The Automounter

For efficiency, the automounter (see NFS Administration Guide) is used to mount FNS directories on demand. The default /etc/auto_master configuration file contains the line:

```
/xfn  -xfn
```

which tells the automounter that the FNS namespace is “mounted” under /xfn, as specified by XFN.
Since the automounter is used to mount directories named through FNS, the subdirectories of an FNS directory cannot be listed until they have been mounted. For example, suppose the file system of the sales organization is composed of multiple NFS file systems. The following `ls` command shows only two file systems that have been visited recently and are currently mounted:

```
% ls /xfn/org/sales/fs
  customers  products
```

To see the entire listing, use the `fnlist(1)` command:

```
% fnlist org/sales/fs
  Listing ‘org/sales/fs’:
  products
goals
customers
incentives
```

### Creating File Contexts

The `fncreate_fs(1M)` command creates file contexts for organizations and sites. It may also be used to override the default file contexts for users and hosts that are created by the `fncreate(1M)` command. See “File Context” on page 82.

There are two methods of using the `fncreate_fs` command. The context bindings may be provided by an input file (See “Creating the Input File” on page 121) or on the command line (See “Using Command-line Input” on page 123).

The two methods of `fncreate_fs` have the following syntax:

```
fncreate_fs [-v] [-r] -f input_file composite_name
fncreate_fs [-v] [-r] composite_name [mount_options] [mount_location...]```
The `fncreate_fs` options `-v` and `-r` are described in Table 8-1.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-v</code></td>
<td>Sets verbose output, displaying information about the contexts being created and modified.</td>
</tr>
<tr>
<td><code>-r</code></td>
<td>Replaces the bindings in the context named by <code>composite_name</code>—and all of its subcontexts—with only those specified in the input. This is equivalent to destroying the context (and, recursively, its subcontexts), and then running <code>fncreate_fs</code> without this option. The <code>-r</code> option should be used with care.</td>
</tr>
</tbody>
</table>

The `fncreate_fs` command manipulates FNS contexts and bindings of the `onc_fn_fs` reference type. It uses an address of type `onc_fn_fs_mount` to represent each remote mount point. The data associated with an address of this type are the corresponding mount options and locations in a single, XDR-encoded string.

Creating the Input File

The input file supplies the names and values to be bound in the context of `composite_name`. Its format is based upon and similar, but not identical, to the format of indirect automount maps (see *NFS Administration Guide*). The input file contains an entry with the form:

```
name [mount_options] [mount_location...]
```

For each entry a reference to the mount locations and the corresponding mount options is bound to the name `composite_name/name`.

The `name` field may be a simple atomic name or a slash-separated hierarchical name. It may also be “.” (dot), in which case the reference is bound directly to `composite_name`.

The `mount_location` field specifies the host or hosts that serve the files for `composite_name/name`. In a simple NFS mount, `mount_location` takes the form:

```
host:path
```
where host is the name of the server from which to mount the file system and path is the path name of the directory to mount.

The mount_options field begins with a hyphen (“−”). This is followed by a comma-separated list (with no spaces) of the mount options to use when mounting the directory. These options also apply to any subcontexts of composite_name/name that do not specify mount options of their own.

If mount_options and mount_location are both omitted, then no reference is bound to composite_name/name. Any existing reference is unbound.

Using the example from Figure 8-1 on page 118, suppose you want jsmith’s file system to be an NFS mount of the directory /export/home/jsmith from host svr1. The command would be run as follows:

```
% fncreate_fs -f infile user/jsmith/fs
```

with infile containing

```
.  svr1:/export/home/jsmith
```

To set up the file system illustrated in Figure 8-2 on page 119, run the command

```
% fncreate_fs -f infile org/engineering/fs
```

with infile containing

```
tools/db   svr1:/export/db
project    svr1:/export/proj
project/src svr2:/export/src
```
To change the NFS mounts for `project` and its subcontext `src` to be read-only, you can change `infile` as follows:

```
tools/db  svr1:/export/db
project -ro  svr1:/export/proj
project/src  svr2:/export/src
```

The `-ro` is unnecessary in the third line. Since `src` is a subcontext of `project`, it will inherit the `-ro` mount option from above.

The following input file would make all of the mounts read-only except for `org/engineering/fs/project/src`.

```
.  -ro
tools/db  svr1:/export/db
project  svr1:/export/proj
project/src  svr2:/export/src
```

Using Command-line Input

The `fncreate_fs(1M)` command also allows the binding description to be provided on the command line:

```
fncreate_fs  composite_name [mount_options] [mount_location ...]
```

This is equivalent to using the first form of the command and providing a one-line input file containing “.” in the `name` field, and the given mount options and locations. The previous example in which `jsmith`’s file system was set could be set from the command line as follows:

```
% fncreate_fs user/jsmith/fs svr1:/export/home/jsmith
```
Similarly, the hierarchy in Figure 8-2 on page 119 could have been set up by running the sequence of commands:

```
% fncreate_fs org/engineering/fs/tools/db svr1:/export/db
% fncreate_fs org/engineering/fs/project svr1:/export/proj
% fncreate_fs org/engineering/fs/project/src svr2:/export/src
```

To make all three of the mounts read-only, you would run this command:

```
% fncreate_fs org/engineering/fs -ro
```

**Advanced Input Formats**

The following two sections apply to both input file and command-line input formats.

**Multiple Mount Locations**

Multiple `mount_location` fields may be specified for NFS file systems that are exported from multiple, functionally equivalent locations:

```
% fncreate_fs org/sales/fs svr1:/sales svr2:/sales
```

The automounter will attempt to choose the best server from among the alternatives provided. If several locations in the list share the same path name, they may be combined using a comma-separated list of host names:

```
% fncreate_fs org/sales/fs svr1,svr2:/sales
```

The hosts may be weighted, with the weighting factor appended to the host name as a nonnegative integer in parentheses: the lower the number, the more desirable the server. The default weighting factor is zero (most desirable).
The following example illustrates one way to indicate that svr2 is the preferred server:

```
% fncreate_fs org/sales/fs svr1(2),svr2(1):/sales
```

See NFS Administration Guide for additional information on how the automounter interprets the `mount_location` field.

**Variable Substitution**

Variable names, prefixed by `$`, may be used in the `mount_options` or `mount_location` fields of `fncreate_fs`. For example, a mount location may be given as

```
svr1:/export/$CPU
```

The automounter will substitute client-specific values for these variables when mounting the corresponding file systems. In the above example, `$CPU` is replaced by the output of `uname -p`; for example, `sparc`.

See NFS Administration Guide for additional information on how the automounter treats variables substitution.

**Backward Compatibility Input Format**

For additional compatibility with automount maps, the following input file format is also accepted by `fncreate_fs`:

```
name [mount_options] [mount_location ...] \ 
/offset1 [mount_options1] mount_location1 ... \ 
/offset2 [mount_options2] mount_location2 ... \ 
...```

where each offset field is a slash-separated hierarchy. The backslash (\) indicates the continuation of a single long line. This is interpreted as being equivalent to

<table>
<thead>
<tr>
<th>name</th>
<th>[mount_options]</th>
<th>[mount_location ...]</th>
</tr>
</thead>
<tbody>
<tr>
<td>name/offset1</td>
<td>[mount_options1]</td>
<td>mount_location1 ...</td>
</tr>
<tr>
<td>name/offset2</td>
<td>[mount_options2]</td>
<td>mount_location2 ...</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The first line is omitted if both mount_options and mount_location are omitted. This format is for compatibility only. It provides no additional functionality, and its use is discouraged.

**Administering File Contexts**

File contexts may be inspected using the fnlist(1) and fnlookup(1) commands, and may be pruned or destroyed using fnunbind(1) and fndestroy(1M). These commands and sample output are described in Chapter 6, “Administering FNS on NIS+.” Refer also to the man page for each command.
Administering the Printer Namespace

This chapter describes the administration of the printer namespace. The printer context is not part of the XFN policies. It is provided in FNS in order to store printer bindings.

The Printer Namespace

FNS provides the capability to store printer bindings in the FNS namespace. This gives print servers the means to advertise their services and allow users to browse and choose amongst the available printers without client side administration.

Printer bindings are stored in printer contexts, which are associated with organizations, users, hosts, and sites. Hence, each organization, user, host, and site has its own printer context.

The printer context is created under the service context of the respective composite name. For example, the composite name shown below has the following printer context:

```
org/wiz.com./service/printer
```
The name of a printer for a host, labpc, with a printer context might look like this:

```
host/labpc/service/printer/laser
```

**Administering Printer Contexts**

Currently, printer contexts are supported for name service of files, NIS, and NIS+. The manner in which the bindings are stored in the printer context varies according to the underlying name service used for implementing FNS. For NIS and files, printer bindings are only associated with organizations and all the bindings exist in one printer context. NIS+, however, stores the printer bindings in the printer context, which allows the printer namespace to be arranged hierarchically and be associated with the org, host, user, and site contexts.

**Using Files**

Files are used as the default name service if neither NIS nor NIS+ is present. The printer bindings are stored in the `/etc/printers.conf` file, which is the printer configuration database used to describe printers. Each printer binding requires its own entry in this file. For example, if you have a printer named printer1, with the alias ps, you would add an entry to the `printers.conf` file in this format:

```
printer1|ps:bsdaddr=server_name,printer_name
```

In this example, when a lookup is performed on an address containing printer1, the address type of `onc_printers_bsdaddr` is returned. For more information about the required file format in `printers.conf`, see the `printers.conf(4)` man page.
Using NIS

If NIS is the underlying name service, the NIS map that is used to store the printer configuration is called `printers.conf.byname`. Each printer binding has an entry in this file. For example, if you have a printer named `printer2`, with the alias `lp`, you would add an entry to the `printers.conf.byname` file in this format:

```
printer2|lp:bsdaddr=server_name,printer_name
```

For more information about the syntax required, see the `printers.conf.byname(4)` man page.

When you list or look up the available printers (using `lpstat(1)`, `fnlist(3N)` or `fn_ctx_lookup()` for example), the results are created by merging the list of printers included in the files name service (`/etc/printers.conf`) with the list of printers included in the NIS name service map (`/etc/printers.conf.byname`).

Using NIS+

If NIS+ is the underlying name service for FNS, administering printer contexts is simplified by the `fncreate_printer` command, which creates the printer context for organization, users, hosts, and sites.

The `fncreate_printer` command takes the following arguments:

```
fncreate_printer composite_name printer_name printer_address
```

where `printer_address` is in the form `address_type=address`. In the next example, the `printer_address` is `bsdaddr=labpc,laser-jet`. For more information, see the `fncreate_printer(1)` man page.

In this example, a printer binding for the printer `laser-jet` for the user `jsmith` is created:

```
% fncreate_printer user/jsmith laser-jet bsdaddr=labpc,laser-jet
```
The new binding, user/jsmith/serviceprinter/laser-jet, has the address type onc_printers_bsdaddr, and the address labpc, laser-jet. FNS adds the prefix onc_printers_ to the address type.

In NIS+, it is possible to organize printers hierarchically. For example, printers can be listed under the printer context, as shown by the following commands:

```plaintext
% fncreate_printer org/wiz.com. color/lpq bsdaddr=colorful,lpq
% fncreate_printer org/wiz.com. color/laser bsdaddr=colorprt,laser
% fncreate_printer org/wiz.com. color/inkj bsdaddr=colorjet,inkj
```

The fncreate command added the printer bindings for the printers, lpq, laser, and inkj to the context color present under the printer context. The result looks like this:

```plaintext
org/wiz.com./service/printer/color/lpq
org/wiz.com./service/printer/color/laser
org/wiz.com./service/printer/color/inkj
```

Similarly, color printers, green, red, and blue for user jsmith can be organized as follows:

```plaintext
user/jsmith/service/printer/color/green
user/jsmith/service/printer/color/red
user/jsmith/service/printer/color/blue
```

Printer bindings (contexts) in NIS+ can be removed using the fndestroy command. For example, to remove the printer context in this example, use the command:

```plaintext
% fndestroy user/jsmith/service/printer/laser-jet
```

When you list or look up the available printers (using lpstat(1), fnlist(3N) or fn_ctx_lookup() for example), the results are created by merging the list of printers included in the files name service (/etc/printers.conf) with the list of printers included in the NIS+ tables generated by fncreate(1) or fncreate_printer(1).
Part 4 — Application Programming

These chapters present information for the application developer.

| Interfaces for Writing XFN Applications       | page 133 |
| XFN Composite Names                          | page 159 |
| XFN Programming Examples                     | page 167 |
This chapter describes the client programming interfaces for XFN. Additional information on the XFN interfaces is available in the man pages.

<table>
<thead>
<tr>
<th>Interface</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>The Base Context Interface</td>
<td>135</td>
</tr>
<tr>
<td>The Base Attribute Interface</td>
<td>143</td>
</tr>
<tr>
<td>Status Objects and Status Codes</td>
<td>150</td>
</tr>
<tr>
<td>Parameters Used in the Interface</td>
<td>153</td>
</tr>
<tr>
<td>Parsing Compound Names</td>
<td>156</td>
</tr>
</tbody>
</table>

**XFN Interface Overview**

The XFN client interface consists of the base context interface, the base attribute interface, and a number of supporting interfaces. The base context interface provides the basic operations for naming, such as binding a name to a reference, looking up the reference bound to a name, and unbinding a name. The base attribute interface provides operations to examine and modify attributes associated with named objects. The supporting interfaces contain:

- Operations on the status object and status codes used in the context and attribute operations.
- A number of abstract data types defined to represent objects passed to and returned from the context and attribute operations, such as composite names, references, and attributes.
• A standard model and operations for parsing compound names, whose syntax is specific to a naming system. These are of primary interest to service implementers.

“API Usage Model” on page 22 summarizes how an application typically uses the programming interface.

**Interface Conventions**

The XFN interface is presented in ISO standard C, which is equivalent to ANSI standard C. The symbols defined by the interface are prefixed by `fn` or `FN`, for federated naming.

• The `FN_` prefix is used for both data types and predefined constants. In addition, data types have a `_t` suffix, such as `FN_ref_t`. Predefined constants appear in all-uppercase characters, such as `FN_ID_STRING`.

• The `fn_` prefix is used for function names. Names of functions in the base context interface have the prefix `fn_ctx_`, such as `fn_ctx_lookup`. Names of functions in the base attribute interface have the prefix `fn_attr_`, such as `fn_attr_get`.

**Usage**

The XFN header file must be included in code as shown for compilation.

```c
#include <xfn/xfn.h>
```

The XFN library must be included in the link line as shown.

```c
cc -o program_name file1.c file2.c -lxfn
```

**Abstract Data Types**

Except for `FN_attrvalue_t` and `FN_identifier_t`, the types defined in the interface hide their actual data representation from the client. The client performs every operation on an object of one of these types through a well-defined interface for that data type.
When the client accesses these objects, the client refers to the objects solely through a *handle* to an object. Operations are provided to create objects of each type and to destroy them. The creation operation returns a handle to the new object. The destroy operation releases all resources associated with the object. The only information about this handle revealed to the client is that it is a pointer type. The client cannot assume what this handle points to. In particular, the handle may not point directly to the memory containing the object’s actual state.

The value 0 is defined for all pointer types. The functions that return handles in the interface return the value 0 as an indication of failure. The values 0 and NULL are equivalent.

**Memory–Management Policies**

The following memory–management policies are used for all client interfaces described in this chapter:

- When a function returns a non-*const* pointer to an object, the client “owns” the object. The client may alter the object and is responsible for freeing the space allocated to it when the object is no longer required.

- When a function returns a *const* pointer to an object, the service “owns” the object. The client must neither modify the object in any way, nor free the space allocated to it. If the client needs to control a copy, it must make one for itself.

- When a function takes a non-*const* parameter that is passed by reference, the service “borrows” the object during the period of the function’s execution. It may modify the object during this period, but it does not retain any reference to the object passed in beyond this period.

- When a function takes a *const* parameter that is passed by reference, the service reads but does not modify the object. The service does not keep any reference to the object beyond the period of the function’s execution.

**The Base Context Interface**

This section describes the operations in the base context interface. The interfaces to the objects used in the operations are described in “Parameters Used in the Interface” on page 153.
Names in Context Operations

In most of the operations of the base context interface, the caller supplies a context and a composite name argument. The supplied composite name is always interpreted relative to the supplied context.

The operation may eventually be effected on a different context called the operation's target context. Each operation has an initial resolution phase that conveys the operation to its target context, following which the operation is applied. The effect (but not necessarily the implementation) is that of

- Doing a lookup on that portion of the name that represents the target context, and then
- Invoking the operation on the target context.

The contexts involved only in the resolution phase are called intermediate contexts. Normal resolution of names in context operations always follows XFN links, which are defined in “XFN Links” on page 15.

Requirements for Supporting the Context Operations

The lookup operation fn_ctx_lookup() must be supported by all contexts. Contexts may indicate that they do not support other operations by returning an FN_E_OPERATION_NOT_SUPPORTED status code (see Table 10-3 on page 151).

XFN contexts are required to support the resolution phase of every operation in the base context and attribute interface when involved in the operation as intermediate contexts. That is, each intermediate context must participate in the process of conveying the operation to the target context, even if it does not support that operation itself. For example, not all contexts need allow binding and listing names. However, all contexts must fully support the resolution phase of these operations.

Composite names are passed to an XFN context implementation in a structural form as an ordered sequence of components. When resolving a name the context implementation is responsible for

- Determining which set of leading components it must resolve
- Resolving that portion to a reference
- Returning a status object containing this reference and the portion of the name unresolved
Composite name resolution is further discussed in “Composite Name Resolution” on page 163.

**Status Objects**

In each context operation, the caller supplies an `FN_status_t` parameter. The called function sets this status object as described in “Status Objects and Status Codes” on page 150. All status objects are handled in this manner for each operation in the base context interface; this will not be restated in the individual operation descriptions.

**Getting Context Handles**

All operations on a context require a context handle. There are two ways of obtaining a context handle. If you have a reference, you can use it to construct a context handle. Otherwise, you must call `fn_ctx_handle_from_initial()` to get a handle to the initial context.

**Construct Handle to Initial Context**

```c
FN_ctx_t *fn_ctx_handle_from_initial(FN_status_t *status);
```

This operation returns a handle to the caller's initial context. On successful return, the context handle points to a context containing the bindings described in “Initial Context Bindings for Naming Within the Enterprise” on page 54 and “Initial Context Bindings for Global Naming” on page 62.

**Construct Context Handle From Reference**

```c
FN_ctx_t *fn_ctx_handle_from_ref(  
    const FN_ref_t *ref,  
    FN_status_t *status);
```

This operation returns a handle to an `FN_ctx_t` object given a reference, `ref`, for that context.
Lookup and List Contexts

Lookup

FN_ref_t *fn_ctx_lookup(
        FN_ctx_t *ctx,
        const FN_composite_name_t *name,
        FN_status_t *status);

This operation returns the reference bound to name relative to the context ctx.

List Names

FN_nameslist_t* fn_ctx_list_names(
        FN_ctx_t *ctx,
        const FN_composite_name_t *name,
        FN_status_t *status);

FN_string_t *fn_namelist_next(
        FN_namelist_t *nl,
        FN_status_t *status);

void fn_namelist_destroy(
        FN_namelist_t *nl,
        FN_status_t *status);

This set of operations is used to list the set of names bound in the context named name relative to the context ctx. name must name a context. If the intent is to list the contents of ctx, name should be an empty composite name.

The call to fn_ctx_list_names() initiates the enumeration process for the target context. It returns an FN_nameslist_t object that can be used for the enumeration.

The operation fn_namelist_next() returns the next name in the enumeration identified by nl and updates nl to indicate the state of the enumeration marker. Successive calls to fn_namelist_next() using nl return successive names and further update the state of the enumeration. fn_namelist_next() returns a NULL pointer when the enumeration has been completed.

fn_namelist_destroy() is used to release resources used during the enumeration. This may be invoked at any time to terminate the enumeration.
The names enumerated using the list names operations are not ordered in any way. There is no guaranteed relation between the order in which names are added to a context and the order names are obtained by enumeration. There is no guarantee that any two enumerations will return the names in the same order.

When a name is added to or removed from the context, this may not necessarily invalidate the enumeration handle that the client holds for that context. If the enumeration handle remains valid, the update may or may not be visible to the client.

**List Bindings**

```c
FN_bindinglist_t* fn_ctx_list_bindings(
    FN_ctx_t *ctx,
    const FN_composite_name_t *name,
    FN_status_t *status);
FN_string_t *fn_bindinglist_next(
    FN_bindinglist_t *bl,
    FN_ref_t **ref,
    FN_status_t *status);
void fn_bindinglist_destroy(
    FN_bindinglist_t *bl,
    FN_status_t *status);
```

This set of operations is used to list the set of names and bindings in the context named by `name`, relative to the context `ctx`. `name` must name a context. If the intent is to list the contents of `ctx`, `name` should be an empty composite name.

The semantics of these operations are similar to those for listing names. In addition to a name string being returned, `fn_bindinglist_next()` also returns the reference of the binding for each member of the enumeration.

**Lookup Link**

```c
FN_ref_t *fn_ctx_lookup_link(
    FN_ctx_t *ctx,
    const FN_composite_name_t *name,
    FN_status_t *status);
```

This operation returns the XFN link bound to `name`. The terminal atomic part of `name` must be bound to an XFN link.
The normal fn_ctx_lookup() operation follows all XFN links encountered, including any bound to the terminal atomic part of name. This operation differs from the normal lookup in that when the terminal atomic part of name is an XFN link, this last link is not followed, and the operation returns the link.

**Updating Bindings**

Bindings can be added, overwritten, removed, or renamed.

**Bind**

```c
int fn_ctx_bind(
    FN_ctx_t *ctx,
    const FN_composite_name_t *name,
    const FN_ref_t *ref,
    unsigned int exclusive,
    FN_status_t *status);
```

This operation binds the supplied reference ref to the supplied composite name name, taken relative to ctx. The binding is made in the target context—that named by all but the terminal atomic part of name. The operation binds the terminal atomic name to the supplied reference in the target context. The target context must already exist.

The value of exclusive determines what happens if the terminal atomic part of the name is already bound in the target context. If exclusive is nonzero and name is already bound, the operation fails. If exclusive is zero, the new binding replaces any existing binding.

The value of ref cannot be NULL. If the intent is to reserve a name using the fn_ctx_bind() operation, a reference containing no address should be bound. This reference may be naming service-specific or it may be the conventional NULL reference.

**Unbind**

```c
int fn_ctx_unbind(
    FN_ctx_t *ctx,
    const FN_composite_name_t *name,
    FN_status_t *status);
```

This operation removes the terminal atomic name in name from the target context—that named by all but the terminal atomic part of name.
This operation is successful even if the terminal atomic name was not bound in
target context, but fails if any of the intermediate names are not bound.
fn_ctx_unbind() operations are idempotent.

Rename

int fn_ctx_rename(
    FN_ctx_t *ctx,
    const FN_composite_name_t *oldname,
    const FN_composite_name_t *newname,
    unsigned int exclusive,
    FN_status_t *status);

This operation binds the reference currently bound to oldname, resolved
relative to ctx to newname, and unbinds oldname. The newname is resolved
relative to the target context—that named by all but the terminal atomic part of
oldname.

If exclusive is zero, this operation overwrites any old binding of newname.
If exclusive is nonzero, the operation fails if newname is already bound.

The only restriction that XFN places on newname is that it be resolved relative
to the target context. For example, in some implementations, newname might be
restricted to be a name in the same naming system as the terminal component
of oldname. In another implementation, newname might be restricted to an
atomic name.

Managing Contexts

Contexts can be created or destroyed.

Create Subcontext

FN_ref_t *fn_ctx_create_subcontext(
    FN_ctx_t *ctx,
    const FN_composite_name_t *name,
    FN_status_t *status);

This operation creates a new context of the same type as the target
context—that named by all but the terminal atomic part of name—and binds it
to the composite name name resolved relative to the context ctx, and returns a
reference to the newly created context.
As with the bind operation, the target context must already exist. The new context is created and bound in the target context using the terminal atomic name in name.

The operation fails if the terminal atomic name already exists in the target context.

The new subcontext exports the context interface and is created in the same naming system as the target context. XFN does not specify any further properties of the new subcontext. Other properties of the subcontext are determined by the target context and its naming system.

Destroy Subcontext

```c
int fn_ctx_destroy_subcontext(
    FN_ctx_t *ctx,
    const FN_composite_name_t *name,
    FN_status_t *status);
```

This operation destroys the subcontext named by name, interpreted relative to ctx, and unbinds the name.

As with the unbind operation, the operation succeeds if the terminal atomic name is not bound in the target context—that named by all but the terminal atomic part of name.

Some aspects of this operation are determined by the target context and its naming system. For example, XFN does not specify what happens if the named subcontext is not empty when the operation is invoked.

Other Context Operations

Get Reference to Context

```c
FN_ref_t *fn_ctx_get_ref(
    const FN_ctx_t *ctx,
    FN_status_t *status);
```

This operation returns a reference to the supplied context object.
Get Syntax Attributes of Context

```c
FN_attrset_t *fn_ctx_get_syntax_attrs(
    FN_ctx_t *ctx,
    const FN_composite_name_t *name,
    FN_status_t *status);
```

This operation returns the syntax attributes associated with the context named by `name`, relative to the context `ctx`.

This operation is different from other XFN attribute operations in that these syntax attributes could be obtained directly from the context. Attributes obtained through other XFN attribute operations may not necessarily be associated with the context; they may be associated with the reference of the context, rather than the context itself (see “Relationship to Naming Operations” on page 144).

Destroy Context Handle

```c
void fn_ctx_handle_destroy(FN_ctx_t *ctx);
```

This operation destroys the context handle `ctx` and allows the implementation to free resources associated with the context handle. This operation does not affect the state of the context itself.

The Base Attribute Interface

This section describes the operations in the base attribute interface. The interfaces to the objects used in operations in this interface are described in “Parameters Used in the Interface” on page 153.

XFN Attribute Model

In the XFN attribute model, a set of zero or more attributes can be associated with a named object. Each attribute in the set has a unique attribute identifier, an attribute syntax, and a set of zero or more distinct attribute values. Each attribute value has an opaque data type. The attribute identifier serves as a name for the attribute. The attribute syntax indicates how the attribute values are encoded.
The operations in the base attribute interface may be used to examine and modify the settings of attributes associated with existing named objects. These objects may be contexts or other types of objects. The attribute operations do not create names or remove names in contexts.

The range of support for attribute operations may vary widely. Some naming systems may not support any attribute operations. Other naming systems may support only read operations or operations on attributes whose identifiers are in some fixed set. A naming system may limit attributes to have a single value or may require at least one value. Some naming systems may only associate attributes with context objects, while others may allow associating attributes with noncontext objects.

Typically, attributes of an object are manipulated through operations that operate on a single attribute, such as reading or updating a single attribute. Moreover, the client is typically expected to be able to read all attribute values of a single attribute in one call. However, sometimes there is a requirement to manipulate several attributes of a single object or to obtain individual attribute values of a single attribute from the name service. To address these requirements, two kinds of attribute operations are defined:

- Single-attribute operations
- Multiple-value and multiple-attribute operations

**Relationship to Naming Operations**

An XFN attribute operation may not necessarily be equivalently expressed as an independent fn_ctx_lookup() operation followed by an attribute operation in which the caller supplies the resulting reference and an empty name. The reason is that in some attribute models, attributes are associated with a named object in the context in which the object is named. In others an object's attributes are stored in the object itself. XFN accommodates both these models.

**Note** – Invoking an attribute operation using the target context and the terminal atomic name accesses either the attributes that are associated with the terminal name or the object named by the terminal name—this is dependent upon the underlying attribute model. This document uses the term “attributes associated with a named object” to refer to all of these cases.
XFN does not provide any guarantee about the relationship between the attributes and the reference associated with a given name. Some naming systems may store the reference bound to a name in one or more attributes associated with a name. Attribute operations might affect the information used to construct a reference.

To avoid undefined results, programmers must use the operations in the context interface and not the attribute operations when manipulating references. Applications should avoid the use of specific knowledge about how an XFN context implementation over a particular naming system constructs references.

**Status Objects**

In each attribute operation, the caller supplies an `FN_status_t` parameter. The called function sets this status object as described in “Status Objects and Status Codes” on page 150. All status objects are handled in this manner for each operation in the base attribute interface; this will not be restated in the individual operation descriptions.

**Single-Attribute Operations**

Each of these operations takes as arguments a context and composite name relative to this context and manipulates the attributes associated with the named object. Each operation sets a status object to describe the status of the operation.

**Get Attribute**

```c
FN_attribute_t *fn_attr_get(
    FN_ctx_t *ctx,
    const FN_composite_name_t *name,
    const FN_identifier_t *attribute_id,
    FN_status_t *status);
```

This operation returns the identifier, syntax, and values of a specified attribute, `attribute_id`, for the object named `name` relative to the context `ctx`. If `name` is empty, the attribute associated with `ctx` is returned.
fn_attr_get_values() and its related functions are for getting individual values of an attribute and should be used if the combined size of all the values are expected to be too large to be returned in a single invocation of fn_attr_get().

Modify Attribute

```c
int fn_attr_modify(
    FN_ctx_t *ctx,
    const FN_composite_name_t *name,
    unsigned int mod_op,
    const FN_attribute_t *attr,
    FN_status_t *status);
```

This operation modifies according to `mod_op` the attribute `attr` associated with the object named `name`, relative to `ctx`. If `name` is empty, the attribute associated with `ctx` is modified.

<table>
<thead>
<tr>
<th>Operation Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>FN_ATTR_OP_ADD</td>
<td>Add an attribute with given attribute identifier and set of values. If an attribute with this identifier exists already, replace the set of values with those in the given set. The set of values may be empty if the target naming system permits.</td>
</tr>
<tr>
<td>FN_ATTR_OP_ADD_EXCLUSIVE</td>
<td>Add an attribute with the given attribute identifier and set of values. The operation fails if an attribute with this identifier exists already. The set of values may be empty if the target naming system permits.</td>
</tr>
<tr>
<td>FN_ATTR_OP_ADD_VALUES</td>
<td>Add the given values to those of the given attribute (resulting in the attribute having the union of its prior value set with the set given). Create the attribute if it does not exist already. The set of values may be empty if the target naming system permits.</td>
</tr>
<tr>
<td>FN_ATTR_OP_REMOVE</td>
<td>Remove the attribute with the given attribute identifier and all its values. The operation succeeds even if the attribute does not exist. The values of the attribute supplied with this operation are ignored.</td>
</tr>
<tr>
<td>FN_ATTR_OP_REMOVE_VALUES</td>
<td>Remove the given values from those of the given attribute (resulting in the attribute having the set difference of its prior value set and the set given). This succeeds even if some of the given values are not in the set of values that the attribute has. In naming systems that require an attribute to have at least one value, removing the last value will remove the attribute as well.</td>
</tr>
</tbody>
</table>
Get Attribute Values

This set of operations allows the caller to obtain attribute values associated with a single attribute individually.

```c
FN_valuelist_t *fn_attr_get_values(
    FN_ctx_t *ctx,
    const FN_composite_name_t *name,
    const FN_identifier_t *attribute_id,
    FN_status_t *status);
```

```c
FN_attrvalue_t *fn_valuelist_next(
    FN_valuelist_t *vl,
    FN_identifier_t **attr_syntax,
    FN_status_t *status);
```

```c
void fn_valuelist_destroy(
    FN_valuelist_t *vl,
    FN_status_t *status);
```

This set of operations is used to obtain the set of values of a single attribute, identified by `attribute_id`, associated with `name`, relative to `ctx`. If `name` is empty, the attribute associated with `ctx` are obtained.

This interface should be used instead of `fn_attr_get()` if the combined size of the all the values is expected to be too large to be returned by `fn_attr_get()`.

The operation `fn_attr_get_values()` initiates the enumeration process. It returns a handle to an `FN_valuelist_t` object that can be used for subsequent `fn_valuelist_next()` calls to enumerate the values requested.

The operation `fn_valuelist_next()` returns the next attribute value in the enumeration and updates `vl` to indicate the state of the enumeration.

The operation `fn_valuelist_destroy()` frees the resources associated with the enumeration. This may be invoked at any time in order to terminate the enumeration.

Multiple-Attribute Operations

These operations allow the caller to specify an operation that operates on multiple attributes using one or more calls.
The failure semantics may vary widely across naming systems. In some systems the single function call may comprise multiple individual naming system operations, with no guarantees of atomicity.

**Get Attribute Identifiers**

```c
FN_attrset_t *fn_attr_get_ids(
    FN_ctx_t *ctx,
    const FN_composite_name_t *name,
    FN_status_t *status);
```

This operation gets a list of all the attribute identifiers that are associated with the object named `name` relative to the context `ctx`. If `name` is empty, the attribute identifiers associated with `ctx` are returned.

**Get Multiple Attributes**

```c
FN_multigetlist_t *fn_attr_multiget(
    FN_ctx_t *ctx,
    const FN_composite_name_t *name,
    const FN_attrset_t *attr_ids,
    FN_status_t *status);
```

```c
FN_attribute_t *fn_multigetlist_next(
    FN_multigetlist_t *ml,
    FN_status_t *status);
```

```c
void fn_multigetlist_destroy(
    FN_multigetlist_t *ml,
    FN_status_t *status);
```

This set of operations gets one or more attributes associated with the object named `name` relative to the context `ctx`. If `name` is empty, the attributes associated with `ctx` are returned.

The attributes returned are those specified in `attr_ids`. If the value of `attr_ids` is 0, all attributes associated with the named object are returned. Any attribute values in `attr_ids` provided by the caller are ignored; only the identifiers are relevant for this operation. Each attribute (identifier, syntax, and values) is returned one at a time using an enumeration scheme similar to that for listing a context. `fn_attr_multi_get()` initiates the enumeration process. It returns a handle to an `FN_multigetlist_t` object that can be used for subsequent `fn_multigetlist_next()` calls to enumerate the attributes requested.
The operation `fn_multigetlist_next()` returns the next attribute (identifier, syntax, and values) in the enumeration and updates `ml` to indicate the state of the enumeration. Successive calls to `fn_multigetlist_next()` using `ml` return successive attributes in the enumeration and further update the state of the enumeration.

The operation `fn_multigetlist_destroy()` frees the resources used during the enumeration. This may be invoked at any time to terminate the enumeration.

Implementations are not required to returned all attributes requested by `attr_ids`. Some may choose to return only the attributes found successfully; such implementations may not necessarily return identifiers for attributes that could not be read.

**Modify Multiple Attributes**

```c
int fn_attr_multi_modify(  
    FN_ctx_t *ctx,  
    const FN_composite_name_t *name,  
    const FN_attrmodlist_t *mods,  
    FN_attrmodlist_t **unexecuted_mods,  
    FN_status_t *status);
```

This operation modifies the attributes associated with the object named `name`, relative to `ctx`.

In the `mods` parameter, the caller specifies a sequence of modifications that are to be done in order on the attributes. Each modification in the sequence specifies a modification operation code (shown in Table 10-1 on page 146) and an attribute on which to operate.

If all the modifications were performed successfully, `unexecuted_mods` is a NULL pointer.

If an error is encountered while performing the list of modifications, `status` indicates the type of error and `unexecuted_mods` is set to point to a list of unexecuted modifications. The contents of `unexecuted_mods` do not share any state with `mods`; items in `unexecuted_mods` are copies of items in `mods` and appear in the same order in which they were originally supplied in `mods`. The first operation in `unexecuted_mods` is the first one that failed, and the code in `status` applies to this modification operation in particular. If `status` indicates a failure and a NULL pointer is returned in `unexecuted_mods`, that indicates no modifications were executed.
Status Objects and Status Codes

The result status of operations in the context interface and the attribute interface is encapsulated in an `FN_status_t` object. This object contains information about how the operation completed: whether an error occurred in performing the operation, the nature of the error, and information that helps locate where the error occurred. If the error occurred while resolving an XFN link, the status object contains additional information about that error.

The status object consists of several items of information.

<table>
<thead>
<tr>
<th>Information Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Primary status code</td>
<td>An unsigned int code describing the disposition of the operation.</td>
</tr>
<tr>
<td>Resolved name</td>
<td>In the case of a failure during the resolution phase of the operation, this is the leading portion of the name that was resolved successfully. Resolution may have been successful beyond this point, but the error might not be pinpointed further.</td>
</tr>
<tr>
<td>Resolved reference</td>
<td>The reference to which the resolved name is bound.</td>
</tr>
<tr>
<td>Remaining name</td>
<td>The remaining unresolved portion of the name.</td>
</tr>
<tr>
<td>Diagnostic message</td>
<td>Any diagnostic message returned by the context implementation.</td>
</tr>
<tr>
<td>Link status code</td>
<td>If an error occurs while resolving an XFN link, the primary status code has the value <code>FN_E_LINK_ERROR</code>, and this code describes the error that occurred while resolving the XFN link.</td>
</tr>
<tr>
<td>Resolved link name</td>
<td>In the case of a link error, this contains the resolved portion of the name in the XFN link.</td>
</tr>
<tr>
<td>Resolved link reference</td>
<td>In the case of a link error, this contains the reference to which the resolved link name is bound.</td>
</tr>
<tr>
<td>Remaining link name</td>
<td>In the case of a link error, this contains the remaining resolved portion of the name in the XFN link.</td>
</tr>
<tr>
<td>Link diagnostic message</td>
<td>Any diagnostic message related to the resolution of the link.</td>
</tr>
</tbody>
</table>
Both the primary status code and the link status code are values of type `unsigned int` that are drawn from the same set of meaningful values. XFN reserves the values 0 through 127 for standard meanings. Currently values and interpretations for the codes in Table 10-3 are determined by XFN.

<table>
<thead>
<tr>
<th>Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>FN_SUCCESS</td>
<td>The operation succeeded.</td>
</tr>
<tr>
<td>FN_E_ATTR_NO_PERMISSION</td>
<td>The caller did not have permission to perform the attempted attribute operation.</td>
</tr>
<tr>
<td>FN_E_ATTR_VALUE_REQUIRED</td>
<td>The operation attempted to create an attribute without a value, and the specific naming system does not allow this.</td>
</tr>
<tr>
<td>FN_E_AUTHENTICATION_FAILURE</td>
<td>The identity of the client principal could not be verified.</td>
</tr>
<tr>
<td>FN_E_COMMUNICATION_FAILURE</td>
<td>An error occurred in communicating with one of the contexts involved in the operation.</td>
</tr>
<tr>
<td>FN_E_CONFIGURATION_ERROR</td>
<td>A problem was detected that indicated an error in the installation of the XFN interfaces.</td>
</tr>
<tr>
<td>FN_E_CONTINUE</td>
<td>The operation should be continued using the remaining name and the resolved reference returned in the status.</td>
</tr>
<tr>
<td>FN_E_CTX_NO_PERMISSION</td>
<td>The client did not have permission to perform the operation.</td>
</tr>
<tr>
<td>FN_E_CTX_NOT_EMPTY</td>
<td>Applies only to <code>fn_ctx_destroy_subcontext()</code>. The naming system required that the context be empty before its destruction, and it was not empty.</td>
</tr>
<tr>
<td>FN_E_CTX_UNAVAILABLE</td>
<td>Service could not be obtained from one of the contexts involved in the operation. This may be because the naming system is busy or is not providing service. In some implementations this may not be distinguished from a communication failure.</td>
</tr>
<tr>
<td>FN_E_ILLEGAL_NAME</td>
<td>The name supplied to the operation was not a well-formed composite name, or one of the component names was not well formed according to the syntax of the naming systems involved in its resolution.</td>
</tr>
<tr>
<td>FN_E_INCOMPATIBLE_CODE_SETS</td>
<td>The operation involved character strings of incompatible code sets or the supplied code set is not supported by the implementation.</td>
</tr>
<tr>
<td>FN_E_INSUFFICIENT_RESOURCES</td>
<td>Either the client or one of the involved contexts could not obtain sufficient resources (on memory, file descriptors, communication ports, stable media space, for example) to complete the operation successfully.</td>
</tr>
</tbody>
</table>
Table 10-3 Status Codes (Continued)

<table>
<thead>
<tr>
<th>Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>FN_E_INVALID_ATTR_VALUE</td>
<td>One of the values supplied was not in the appropriate form for the given attribute.</td>
</tr>
<tr>
<td>FN_E_INVALID_ENUM_HANDLE</td>
<td>The enumeration handle supplied was invalid, either because it was from another enumeration, because an update operation occurred during the enumeration, or for some other reason.</td>
</tr>
<tr>
<td>FN_E_INVALID_SYNTAX_ATTRS</td>
<td>The syntax attributes supplied are invalid or insufficient to fully specify the syntax.</td>
</tr>
<tr>
<td>FN_E_LINK_ERROR</td>
<td>An error occurred while resolving an XFN link encountered during resolution of the supplied name.</td>
</tr>
<tr>
<td>FN_E_LINK_LOOP_LIMIT</td>
<td>A nonterminating loop (cycle) in the resolution is suspected. This arises due to XFN links encountered during the resolution of a supplied composite name. This code indicates either the definite detection of such a cycle, or that resolution exceeded an implementation-defined limit on the number of XFN links allowed for a single operation invoked by the caller (and thus a cycle is suspected).</td>
</tr>
<tr>
<td>FN_E_MALFORMED_LINK</td>
<td>A malformed link reference was encountered. For fn_ctx_lookup_link(), the name supplied resolved to a reference that was not a link.</td>
</tr>
<tr>
<td>FN_E_MALFORMED_REFERENCE</td>
<td>A context object could not be constructed from the supplied reference because the reference was not properly formed.</td>
</tr>
<tr>
<td>FN_E_NAME_IN_USE</td>
<td>(Only for operations that bind names.) The supplied name was already in use.</td>
</tr>
<tr>
<td>FN_E_NAME_NOT_FOUND</td>
<td>Resolution of the supplied composite name proceeded to a context in which the next atomic component of the name was not bound.</td>
</tr>
<tr>
<td>FN_E_NO_SUCH_ATTRIBUTE</td>
<td>The object does not have an attribute with the given identifier.</td>
</tr>
<tr>
<td>FN_E_NO_SUPPORTED_ADDRESS</td>
<td>A context object could not be constructed from a particular reference. The reference contained no address type over which the context interface was supported.</td>
</tr>
<tr>
<td>FN_E_NOT_A_CONTEXT</td>
<td>Either one of the intermediate atomic names did not name a context, and resolution could not proceed beyond this point, or the operation required that the caller supply the name of a context, and the name did not resolve to a reference for a context</td>
</tr>
<tr>
<td>FN_E_OPERATION_NOT_SUPPORTED</td>
<td>The operation attempted is not supported.</td>
</tr>
<tr>
<td>FN_E_PARTIAL_RESULT</td>
<td>The operation attempted is returning a partial result.</td>
</tr>
</tbody>
</table>
Parameters Used in the Interface

This section gives an overview of the types of parameters that are passed and returned by operations in the base context and attribute interfaces. Manipulation of these objects using their corresponding interfaces does not affect their representation in the underlying naming system. Changes to objects in the underlying naming system can only be effected through the use of the interfaces described in “The Base Context Interface” on page 135 and “The Base Attribute Interface” on page 143.

Composite Names

A composite name is represented by an object of type FN_composite_name_t. A composite name is a sequence of components, where each component is a string (of type FN_string_t) intended to contain a name from a single naming system. (See “Syntax” on page 159 for a description of composite name syntax and structure.) Operations are provided to iterate over this sequence, modify it, and compare two composite names.

References and Addresses

A reference is represented by the type FN_ref_t. An object of this type contains a reference type and a list of addresses. The ordering in this list at the time of binding might not be preserved when the reference is returned upon lookup.

The reference type is represented by an object of type FN_identifier_t. The reference type is intended to identify the class of object referenced, but XFN does not dictate its precise use.

Table 10-3 Status Codes (Continued)

<table>
<thead>
<tr>
<th>Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>FN_E_SYNTAX_NOT_SUPPORTED</td>
<td>The syntax type specified is not supported.</td>
</tr>
<tr>
<td>FN_E_TOO_MANY_ATTR_VALUES</td>
<td>The operation attempted to associate more values with an attribute than the naming system supported.</td>
</tr>
<tr>
<td>FN_E_UNSPECIFIED_ERROR</td>
<td>An error occurred that could not be classified by any of the other error codes.</td>
</tr>
</tbody>
</table>
Each address in a reference is represented by an object of type \texttt{FN\_ref\_addr\_t}. An address consists of an opaque data buffer and a type field, again of type \texttt{FN\_identifier\_t}. The address type is intended to identify the mechanism that should be used to reach the object using that address. Multiple addresses in a single reference are intended to identify multiple communication endpoints for the same conceptual object. Multiple addresses may arise for various reasons; for example, because the object offers interfaces over more than one communication mechanism.

The client process must interpret the contents of the opaque buffers based on the type of the address and on the type of the reference. However, this interpretation is intended to occur below the application layer. Most applications developers should not have to manipulate the contents of either address or reference objects themselves. These interfaces would generally be used within service libraries.

**Identifiers**

Identifiers are used to identify reference types and address types in the reference and to identify attributes and their syntax in the attribute operations.

The \texttt{FN\_identifier\_t} type is used to represent an identifier. It consists of an unsigned integer, which determines the format of identifier, and the actual identifier, which is expressed as a sequence of octets.

XFN defines a small number of standard forms for identifiers, as shown in Table 10-4.

<table>
<thead>
<tr>
<th>Identifier Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{FN_ID_STRING}</td>
<td>The identifier is an ASCII string (ISO 646).</td>
</tr>
<tr>
<td>\texttt{FN_ID_DCE_UUID}</td>
<td>The identifier is an OSF DCE UUID in string representation. See the X/Open DCE RPC (ISBN 1-872630-95-2).</td>
</tr>
<tr>
<td>\texttt{FN_ID_ISO_OID_STRING}</td>
<td>The identifier is an ISO OID in ASN.1 dot-separated integer list string format. See the ISO ASN.1 (ISO 8824).</td>
</tr>
<tr>
<td>\texttt{FN_ID_ISO_OID_BER}</td>
<td>The identifier is an ISO OID in ASN.1 Basic Encoding Rules (BER) format. See the ISO BER (ISO 8825).</td>
</tr>
</tbody>
</table>
Strings

The FN_string_t type is used to represent character strings in the XFN interface. It provides a layer of insulation from specific string representations. The FN_string_t operations contain operations for string comparison, substring searches, and manipulation. The FN_string_t type supports multiple code sets. In Solaris 2.5, FNS supports ISO 646.

Attributes and Attribute Values

An attribute has an attribute identifier, a syntax, and a set of distinct values. An attribute is represented by the FN_attribute_t type. The attribute identifier and its syntax are specified using an FN_identifier_t. Each value is a sequence of octets, represented by the FN_attrvalue_t type.

There are operations to allow the construction, destruction, and manipulation of an attribute.

Attribute Sets

An attribute set is a set of attribute objects with distinct attribute identifiers. Attribute sets are represented by the FN_attrset_t type.

There are operations to allow the construction, destruction, and manipulation of an attribute set.

Attribute-Modification Lists

An attribute-modification list allows you to specify multiple modification operations to be performed on the attributes associated with a single named object. An attribute-modification list is represented by the FN_attrmodlist_t type. It consists of an ordered list of attribute-modification specifiers. An attribute-modification specifier consists of an operation and an attribute object. The attribute’s identifier indicates the attribute that is to be operated upon. The attribute’s values are used in a manner depending on the operation. The operation specifier is one of the values described in Table 10-1 on page 146. The operations are to be done in the order in which they appear in the list.
Parsing Compound Names

Most applications treat names as opaque data; therefore, the majority of clients of the XFN interface will not need to parse compound names from specific naming systems. Some applications, however, such as browsers, need such capabilities. For these applications, XFN provides support in the form of the \texttt{FN\_compound\_name\_t} object.

Syntax Attributes

Each context has an associated set of syntax-related attributes. The attribute \texttt{fn\_syntax\_type (FN\_ID\_STRING format)} identifies the naming syntax supported by the context. The value “standard” (ASCII attribute syntax) in the \texttt{fn\_syntax\_type} attribute specifies that the context supports the XFN standard syntax model that is by default supported by the \texttt{FN\_compound\_name\_t} object.

Implementations may choose to support other syntax types in addition to or in place of the XFN standard syntax model, in which case the value of the \texttt{fn\_syntax\_type} attribute would be set to an implementation-specific string and different or additional syntax attributes would be in the set.

Syntax attributes of a context may be generated automatically by a context, in response to \texttt{fn\_ctx\_get\_syntax\_attrs()}, or may be created and updated using the attribute operations. This is implementation dependent.

XFN Standard Syntax Model

Each naming system in an XFN federation has a naming convention. XFN defines a standard model of expressing compound name syntax that covers a large number of specific name syntaxes. This model is expressed in terms of syntax properties of the naming convention and it uses XFN attributes to describe properties of the syntax.

Unless otherwise qualified, the syntax attributes described in this section have attribute identifiers that use the \texttt{FN\_ID\_STRING} format. This does not specify or restrict the use of other formats for identifiers of additional syntax attributes supported by specific implementations.

In the XFN standard syntax model these attributes are interpreted according to the following rules:
• In a string without quotes or escapes, any instance of the separator string delimits two atomic names.

• A separator, quotation mark, or escape string is escaped if preceded immediately (on the left) by the escape string.

• A non-escaped begin-quote that precedes a component must be matched by a non-escaped end-quote at the end of the component. Quotes embedded in nonquoted names are treated as simple characters and do not need to be matched. An unmatched quotation fails with the status code FN_E_ILLEGAL_NAME.

• If there are multiple values for begin-quote and end-quote, a specific begin-quote value must be matched with its corresponding end-quote value.

• When the separator appears between a (non-escaped) begin-quote and the end-quote, it is ignored.

• When the separator is escaped, it is not treated as a separator. An escaped begin-quote or end-quote string is not treated as a quotation mark. An escaped escape string is not treated as an escape string.

• A non-escaped escape string appearing within quotes is interpreted as an escape string. This can be used to embed an end-quote within a quoted string.

After constructing a compound name from a string, the resulting component atoms have one level of escape strings and quotations interpreted and consumed.

Code set mismatches that occur during the construction of the compound name’s string form are resolved in an implementation-dependent way. When an implementation discovers that a compound name has components with incompatible code sets, it returns the error code FN_E_INCOMPATIBLE_CODE_SETS.
Table 10-5 lists all the XFN standard syntax model attributes.

Table 10-5  XFN Syntax Attributes

<table>
<thead>
<tr>
<th>Attribute Identifier</th>
<th>Attribute Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>fn_syntax_type</td>
<td>Its value is the ASCII string “standard” if the context supports the XFN standard syntax model. Its value is an implementation-specific value if another syntax model is supported.</td>
</tr>
<tr>
<td>fn_syntax_direction</td>
<td>Its value is an ASCII string, one of “left-to-right,” “right-to-left,” or “flat.” This determines whether the order of components in a compound name string goes from left-to-right, right-to-left, or whether the namespace is flat (that is, not hierarchical, with all names atomic)</td>
</tr>
<tr>
<td>fn_std_syntax_separator</td>
<td>Its value is the separator string for this name syntax. This attribute is required unless the fn_syntax_direction is flat.</td>
</tr>
<tr>
<td>fn_std_syntax_escape</td>
<td>If present, its value is the escape string for this name syntax.</td>
</tr>
<tr>
<td>fn_std_syntax_case_insensitive</td>
<td>If present, it indicates that names that differ only in case are considered identical. If this attribute is absent, it indicates that case is significant. If a value is present, it is ignored.</td>
</tr>
<tr>
<td>fn_std_syntax_begin_quote</td>
<td>If present, its value is the begin-quote string for this syntax.</td>
</tr>
<tr>
<td>fn_std_syntax_end_quote</td>
<td>If present, its value is the end-quote string for this syntax.</td>
</tr>
<tr>
<td>fn_std_syntax_ava_separator</td>
<td>If present, its value is the attribute-value assertion separator string for this syntax.</td>
</tr>
<tr>
<td>fn_std_syntax_typeval_separator</td>
<td>If present, its value is the attribute type-value separator string for this syntax.</td>
</tr>
<tr>
<td>fn_std_syntax_code_sets</td>
<td>If present, its value identifies the code sets of the string representation for this syntax. Its value consists of a structure containing an array of code sets supported by the context; the first member of the array is the preferred code set of the context. The values for the code sets are defined in the X/Open code set registry currently defined in DCE RFC 40.1. If this attribute is not present, or if the value is empty, the default code set is ISO 646 (same encoding as ASCII).</td>
</tr>
<tr>
<td>fn_std_syntax_locale_info</td>
<td>If present, its value identifies locale information, such as character set information, of the string representation for this syntax. The interpretation of its value is implementation dependent.</td>
</tr>
</tbody>
</table>
This chapter describes XFN composite names in detail.

<table>
<thead>
<tr>
<th>Syntax</th>
<th>page 159</th>
</tr>
</thead>
<tbody>
<tr>
<td>Composite Name and Naming System Boundaries</td>
<td>page 161</td>
</tr>
<tr>
<td>Composite Name Resolution</td>
<td>page 163</td>
</tr>
<tr>
<td>Strong Separation</td>
<td>page 161</td>
</tr>
<tr>
<td>Weak Separation</td>
<td>page 162</td>
</tr>
<tr>
<td>Explicit NNSPs: Junctions</td>
<td>page 163</td>
</tr>
<tr>
<td>Implicit NNSPs</td>
<td>page 164</td>
</tr>
<tr>
<td>XFN Links</td>
<td>page 165</td>
</tr>
</tbody>
</table>

**Syntax**

The standard string form for XFN composite names is the concatenation of the components of a composite name from left to right, with the XFN component separator character (/) separating each component. Components can be quoted using either double-quote (""") or single-quote (‘’) pairs. The XFN component separator or quote characters may be escaped using a backslash character (\) if the intention is for these characters not to behave as separators or quotes. Note that quotation marks and escape characters are interpreted as such only when they appear in places that need quotes or escapes. For example, a quote appearing in an unquoted component is not interpreted as a quote.
XFN defines an abstract data type, `FN_composite_name_t`, for representing the structural form of a composite name. XFN also defines the syntax of how component string names are composed into an XFN composite name and the corresponding rules for converting an XFN composite name to its structural form from its string form, and vice versa. The XFN client interface includes operations that perform these conversions.

Table 11-1 contains some examples of how the string form of XFN composite names are decomposed into components according to the syntax of XFN composite names. See also Appendix A, “XFN Composite Names Syntax,” for more information.

**Table 11-1 String and Structural Forms of XFN Composite Names**

<table>
<thead>
<tr>
<th>Stringform</th>
<th>Components in <code>FN_composite_name_t</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>a</td>
</tr>
<tr>
<td>a/b/c</td>
<td>a, b, c</td>
</tr>
<tr>
<td>a/</td>
<td>a, &quot;&quot;</td>
</tr>
<tr>
<td>/a</td>
<td>&quot;&quot;, a</td>
</tr>
<tr>
<td>a//</td>
<td>a, &quot;&quot;, &quot;&quot;</td>
</tr>
<tr>
<td>a//b</td>
<td>a, &quot;&quot;, b</td>
</tr>
<tr>
<td>&quot;&quot;</td>
<td>&quot;&quot;</td>
</tr>
<tr>
<td>/</td>
<td>&quot;&quot;, &quot;&quot;</td>
</tr>
<tr>
<td>//</td>
<td>&quot;&quot;, &quot;&quot;, &quot;&quot;</td>
</tr>
<tr>
<td>&quot;a/b/c&quot;/d</td>
<td>a/b/c, d</td>
</tr>
<tr>
<td>&quot;a.b.c&quot;/d</td>
<td>a.b.c, d</td>
</tr>
<tr>
<td>a.b.c/d</td>
<td>a.b.c, d</td>
</tr>
<tr>
<td>a&quot;b/c</td>
<td>a&quot;b, c</td>
</tr>
<tr>
<td>a'b/c</td>
<td>a'b, c</td>
</tr>
<tr>
<td>&quot;a/b/c</td>
<td>illegal name</td>
</tr>
<tr>
<td>&quot;\a/b/c</td>
<td>&quot;a, b, c</td>
</tr>
<tr>
<td>a\b\c/d</td>
<td>a\b\c, d</td>
</tr>
<tr>
<td>a\b\c</td>
<td>a\b\c</td>
</tr>
</tbody>
</table>
Composite Name and Naming System Boundaries

There may not be a one-to-one correspondence between component separators and naming system boundaries if a composite name contains names from naming systems that use the same character as the XFN component separator to separate their atomic names. Consequently, a component of a composite name may represent an atomic name from a hierarchical naming system that uses the XFN component separator or a compound name. **Strong separation** and **weak separation** refer to whether a context always treats the XFN component separator as a naming system boundary.

**Strong Separation**

An XFN context that treats the XFN component separator as a naming system boundary supports strong separation. An XFN component separator that appears within a component to be resolved by the context must be escaped or quoted.

Support for strong separation is a property of a context. A context that supports strong separation expects to receive the name that it is going to resolve entirely in one component of the composite name structure. When a composite name is supplied to such a context, it consumes the leading component of the name; any remaining components are left to be resolved by subordinate naming systems.

An XFN context with a name syntax that is either flat or hierarchical, and does not use the XFN component separator as its atomic separator, supports strong separation. Examples of naming systems that support strong separation are
DNS and NIS+, both of which have right-to-left dot-separated names. The following are examples of names with DNS and NIS+ components, respectively.

```
.../wiz.com/orgunit/ppt
orgunit/accounts payable.finance/user/jsmith
```

**Weak Separation**

An XFN context that does not always treat the XFN component separator as a naming system boundary supports weak separation. This arises when the component naming system associated with the context uses the same character as the XFN component separator as its atomic component separator, and the context allows its atomic separator to appear unescaped and unquoted in its compound names when they occur in composite names. This means that an XFN component separator may not necessarily signify a naming system boundary.

Support for weak separation is a property of a context. A context that supports weak separation expects to receive its atomic names in separate components of the composite name structure. When a composite name is supplied to a context that supports weak separation, the context consumes the leading components of the name (and treats them as atomic components); any remaining components are resolved by subordinate naming systems. The number of components consumed is determined either syntactically or dynamically.

CDS names and X.500 names are examples of names that use the XFN component separator as their atomic name separator. X.500 supports weak separation using a syntactic method (by scanning for typed names) while CDS supports weak separation by determining the naming system boundary dynamically.

The following example shows a composite name with an X.500 component.

```
.../c=us/o=wiz.com/orgunit/ppt
```

**Note** – An XFN context that supports weak separation using only syntax-specific discovery of its naming system boundary may not always be able to be federated with arbitrary subordinate naming systems. If the subordinate
naming system has a naming syntax that is indistinguishable from that of the 
superior naming system, the superior naming system would not be able to 
identify the naming system boundary.

Naming systems that use the same character as the XFN component separator 
as their atomic separator, and which cannot support weak separation because 
it cannot use a syntactic or dynamic method to determine the naming system 
boundary, must provide context implementations that support strong 
separation. This means that occurrences of atomic separators must be quoted 
or escaped when they appear in compound names within composite names.

**Composite Name Resolution**

Composite name resolution combines resolution in each component naming 
system and resolution across federated naming system boundaries. There are 
several techniques for resolving an XFN composite name in the underlying 
federation of naming systems.

This section describes two implementation techniques for composite name 
resolution across a naming system boundary. One technique uses an *explicit* 
next naming system pointer (NNSP) to resolve across a naming system 
boundary. The other uses an *implicit* NNSP to resolve across a naming system 
boundary.

An NNSP is the XFN reference of an XFN context in which composite name 
components from subordinate naming systems are to be resolved. NNSPs are 
entities that “tie” naming systems together into a federated system. NNSPs can 
be bound to names, in which case they are *explicit* NNSPs or *junctions*. NNSPs 
can also be nameless, in which case they are *implicit* NNSPs.

**Explicit NNSPs: Junctions**

A junction is an atomic name that is bound to an NNSP. It is a terminal name 
in the superior naming system. There is no limit on the number of junctions 
bound in a single context, except that imposed by the context. A context may 
reserve certain names for use as junctions or have other policies for selecting 
names for use as junction. The conventions used for identifying junctions and 
their references are context-specific.
Composite name resolution involving junctions proceeds as follows, depending on whether the context supports strong or weak separation.

A context that supports strong separation and junctions consumes the first component of the composite name supplied to it. The last atomic name of the first component must be a junction. Any remaining components are resolved in the context named by the junction.

A context that supports weak separation and junctions resolves a composite name by consuming leading components until a junction is reached, at which point resolution of any remaining components is continued in the context resolved to by the junction. Determination of whether a component is a junction can be done statically using a syntactic policy or dynamically during resolution.

**Implicit NNSPs**

When a context does not want to use part of its namespace for junctions, it uses implicit NNSPs for federating subordinate naming systems. An implicit NNSP is named using the XFN component separator. For example, the name `wiz.com/` names the implicit NNSP of `wiz.com`. Each context can have one implicit NNSP.

Composite name resolution involving implicit NNSPs proceeds as follows, depending on whether the context supports strong or weak separation.

A context that supports strong separation and resolves composite names using an implicit NNSP consumes the first component of the composite name supplied to it. Any remaining components are resolved in the context pointed to by the implicit NNSP of the first component.

A context that supports weak separation and implicit NNSPs in its implementation needs to distinguish the use of the XFN component separator character as an XFN component separator or an atomic separator. This means that such a context needs to know when to exit the current (native) naming system and follow the NNSP. This can be achieved using a static, syntactic policy or a dynamic, resolution-based policy.

With the syntactic policy, a context syntactically discovers where the boundary between its naming system and the subordinate naming system lies. This may impose certain restrictions on the syntax of subordinate naming systems. Subordinate naming systems must not permit as valid top-level names that are
syntactically indistinguishable from names allowed in the superior naming system. For example, assume the superior naming system has a name syntax whose distinguishing feature is that each atomic part must have an equal sign (=). The superior naming system might impose as a policy that subordinate naming systems must not have top-level names that have an equal sign in them. Resolution in the superior naming system continues until all leading components of the supplied composite name fitting the syntactic rule are consumed. Any remaining components are resolved in the context of the NNSP of the last component fitting the syntactic rule.

If a context is not able to syntactically differentiate between atomic components and composite name components, or does not want to impose any syntactic restrictions, it may be able to determine the naming system boundary at runtime during resolution. The policy is to continue resolution in the current naming system until resolution fails, at which point the implicit NNSP associated with the last context at which resolution succeeded is used to continue the resolution. A conflict arises if the same atomic name is bound both in the last context and the context pointed to by the last context’s implicit NNSP. In this case, the binding in the last context takes precedence. Note that this way of supporting weak separation requires the context to have the capability of returning remaining unresolved parts of a given name.

**Coexistence of Explicit and Implicit NNSPs**

Naming systems that implement either technique may coexist in a federation. A naming system that supports composite name resolution using junctions can be federated with one that supports implicit NNSPs, and vice versa.

**XFN Links**

An XFN link affects name resolution in the following way. Suppose \( \text{name} \) is a link bound to the atomic name \( \text{aname} \) in the context \( \text{ctx} \). If at some point resolution of a composite name \( \text{cname} \) reaches the context \( \text{ctx} \) and the next atomic name is \( \text{aname} \), resolution of \( \text{aname} \) results in the resolution of the link \( \text{name} \). This is termed “following the link.” If the first component of the link \( \text{name} \) is the atomic name “,” the remaining components of \( \text{name} \) are resolved relative to \( \text{ctx} \); otherwise, \( \text{name} \) is resolved from the initial context. The resolution of any remaining portion of the name \( \text{cname} \) proceeds from the reference that results by resolving \( \text{name} \).
The link name may itself cause resolution to resolve through other links. This gives rise to the possibility of a cycle of links whose resolution could not terminate normally. As a simple means to avoid such nonterminating resolutions, implementations may define limits on the number of XFN links that may be resolved in any single operation invoked by the caller.
This chapter presents self-contained executable programs for a namespace browser and a printer client and server.
Namespace Browser Example

The first example is a browser that lists all names that it finds in the namespace. When the program is invoked, the browser is set at the initial context or the composite name given on the command line.

Figure 12-1 illustrates the XFN APIs that are used by the browser application. See “Commands” on page 175” and “Sample Output” on page 176”.

Code Example 12-1 fnbrowse Source Code

```c
/*
 * fnbrowse.c -- FNS namespace browser.
 *
 * To keep this example program relatively short, limited error
 * checking is done.
 */
```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <xfn/xfn.h>

#define LINELEN 128 /* maximum length of input line */
typedef enum {CMD_DOWN, CMD_UP, CMD_LIST, CMD_SHOW, CMD_QUIT} command;

FN_status_t *status;

/* Look up a context named relative to the initial context. */
FN_ctx_t *lookup(const FN_composite_name_t *name);

/* Set the browser's focus to the given context. */
void browse(FN_ctx_t *ctx);

/* Set the browser's focus to a subcontext of the given context. */
void cmd_down(FN_ctx_t *ctx, const FN_composite_name_t *child);

/* Print the names bound within a context. */
void cmd_list(FN_ctx_t *ctx);

/*
 * Print a description of the reference bound to "child" in the given
 * context or, if "child" is the empty string, the reference of the
 * context itself.
 */
void cmd_show(FN_ctx_t *ctx, const FN_composite_name_t *child);

/*
 * Read and parse the next command typed by the user. If the command
 * has an argument, set *argp to point to the argument.
 */
command read_command(FN_string_t **argp);

/* Print an error message and the status description */
void error(const char *msg);
int main(int argc, char *argv[]) {
    FN_string_t *arg;

    switch (argc) {
        case 1:
            arg = fn_string_create();
            break;
        case 2:
            arg = fn_string_from_str((unsigned char *)argv[1]);
            break;
        default:
            fprintf(stderr, "Usage: %s [<composite_name>]\n", argv[0]);
            return (1);
    }
    status = fn_status_create();
    browse(lookup(fn_composite_name_from_string(arg)));
    return (0);
}

FN_ctx_t * lookup(const FN_composite_name_t *name) {
    FN_ctx_t *ctx;
    FN_ref_t *ref;
    ctx = fn_ctx_handle_from_initial(status);
    if (ctx == NULL) {
        error("Could not construct initial context");
        exit(1);
    }
    if (fn_composite_name_is_empty(name)) {
        return (ctx);
    }
    ref = fn_ctx_lookup(ctx, name, status);
    fn_ctx_handle_destroy(ctx);
    if (ref == NULL) {
        error("Lookup failed");
        exit(1);
    }
}
ctx = fn_ctx_handle_from_ref(ref, status);
fn_ref_destroy(ref);
if (ctx == NULL) {
    error("Could not construct context handle");
    exit(1);
}
return (ctx);

void
browse(FN_ctx_t *ctx)
{
    FN_string_t *arg;
    FN_composite_name_t*child;

    while (1) {
        switch (read_command(&arg)) {
            case CMD_DOWN:
                child = fn_composite_name_from_string(arg);
                fn_string_destroy(arg);
                cmd_down(ctx, child);
                fn_composite_name_destroy(child);
                break;
            case CMD_UP:
                return;
            case CMD_LIST:
                cmd_list(ctx);
                break;
            case CMD_SHOW:
                child = fn_composite_name_from_string(arg);
                fn_string_destroy(arg);
                cmd_show(ctx, child);
                fn_composite_name_destroy(child);
                break;
            case CMD_QUIT:
                exit(0);
                break;
        }
    }
}
cmd_down(FN_ctx_t *ctx, const FN_composite_name_t *child)
{
    FN_ref_t *ref;
    FN_ctx_t *subctx;

    ref = fn_ctx_lookup(ctx, child, status);
    if (ref == NULL) {
        error("Lookup failed");
        return;
    }
    subctx = fn_ctx_handle_from_ref(ref, status);
    fn_ref_destroy(ref);
    if (subctx == NULL) {
        error("Could not construct context handle");
        return;
    }
    browse(subctx);
    fn_ctx_handle_destroy(subctx);
}

void
cmd_list(FN_ctx_t *ctx)
{
    FN_string_t *empty_string = fn_string_create();
    FN_composite_name_t *empty_name;
    FN_namelist_t *children;
    FN_string_t *child;
    unsigned int statcode;
    int has_children = 0;

    empty_name = fn_composite_name_from_string(empty_string);
    fn_string_destroy(empty_string);

    children = fn_ctx_list_names(ctx, empty_name, status);
    fn_composite_name_destroy(empty_name);

    if (children == NULL) {
        error("Could not list names");
        return;
    }
while ((child = fn_namelist_next(children, status)) != NULL) {
    has_children = 1;
    printf("%s ", fn_string_str(child, &statcode));
    fn_string_destroy(child);
}
if (has_children) {
    printf("\n");
}
fn_namelist_destroy(children, status);

void
cmd_show(FN_ctx_t *ctx, const FN_composite_name_t *child)
{
    FN_string_t *desc;
    FN_ref_t *ref;
    unsigned int statcode;

    ref = fn_ctx_lookup(ctx, child, status);
    if (ref == NULL) {
        error("Lookup failed");
        return;
    }

    desc = fn_ref_description(ref, 2, NULL);
    fn_ref_destroy(ref);
    if (desc != NULL) {
        printf("%s", fn_string_str(desc, &statcode));
        fn_string_destroy(desc);
    } else {
        printf("[No description]\n");
    }
}

command
read_command(FN_string_t **argp)
{
    char buf[LINELEN + 1];
    char *cmd;
    char *child;
while (printf("\n> "), fflush(stdout), gets(buf) != NULL) {
    cmd = strtok(buf, " \	");
    if (cmd == NULL) {
        continue;
    }
    if (strcmp(cmd, "down") == 0) {
        child = strtok(NULL, " \	");
        if (child != NULL) {
            *argp = fn_string_from_str((unsigned char *)child);
            return (CMD_DOWN);
        }
    }
    if (strcmp(cmd, "up") == 0) {
        return (CMD_UP);
    }
    if (strcmp(cmd, "list") == 0) {
        return (CMD_LIST);
    }
    if (strcmp(cmd, "show") == 0) {
        child = strtok(NULL, " \	");
        *argp = (child != NULL)
            ? fn_string_from_str((unsigned char *)child)
            : fn_string_create();
        return (CMD_SHOW);
    }
    if (strcmp(cmd, "quit") == 0) {
        return (CMD_QUIT);
    }
    fprintf(stderr, "Valid commands are:  
    "down <child>, up, list, show [<child>], quit\n");
    return (CMD_QUIT); /* EOF */
}

void
error(const char *msg)
{
    FN_string_t *reason;
    unsigned int statcode;
    fprintf(stderr, "%s", msg);
    reason = fn_status_description(status, 0, NULL);
    if (reason != NULL) {
        fprintf(stderr, ": %s",

Compiling and Executing Browser Example

To compile Code Example 12-1, type:

```
% cc -o fnbrowse fnbrowse.c -lxfn
```

To browse the namespace starting from the initial context, the program is invoked as

```
% fnbrowse
```

Or to browse a composite name and its descendents, type

```
% fnbrowse composite_name
```

Commands

The commands supported by the fnbrowse program are summarized in Table 12-1.

<table>
<thead>
<tr>
<th>Command</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>down child</td>
<td>Sets the browser at the subcontext of the child</td>
</tr>
<tr>
<td>up</td>
<td>Sets the browser at one level higher than the current context</td>
</tr>
<tr>
<td>list</td>
<td>Lists the names bound within the current context</td>
</tr>
<tr>
<td>show</td>
<td>Prints the reference of the current context</td>
</tr>
<tr>
<td>show child</td>
<td>Prints the reference of the current context’s child</td>
</tr>
<tr>
<td>quit</td>
<td>Exits the browser</td>
</tr>
</tbody>
</table>
Sample Output

Sample output for navigating the entire namespace is displayed here.

Note the following:

- The first `list` command shows the initial context bindings.
- The `fnbrowse` program lists all names it finds in the namespace, including names with underscores. These names are explained in “Initial Context Bindings for Naming Within the Enterprise” on page 54.
- The three dots (…) represent the global namespace.

```bash
% fnbrowse
> list
  _myorgunit  ...  _myself thishost myself _orgunit _host
  _thisens myens thisens org orgunit thisuser _thishost
  myorgunit _user thisorgunit host _thisorgunit _myens user
```

Navigating the namespace is accomplished with the `up` and `down` commands. In the following output, the `down` command brings the focus of the browser to the enterprise root of the namespace, `thisens` (can also be `myens`). The `show` command displays information about the reference and address type for `thisens`.

```bash
> down thisens
> show
  Reference type: onc_fn_enterprise
  Address type: on_fn_nisplus
    length: 20
    context type: enterprise root
    representation: normal
    version: 0
    internal name: eng.wiz.com

> up
> down thisorgunit
```
Continuing with the example, this `list` command shows the contexts for this orgunit.

```
> list
service _fs _host _service _site site _user host fs user

> down usr
Lookup failed: Name Not Found: 'usr'

> down service
> list
printer

> down printer
```

The `list` command shows the printer names that are bound in the `printer` context. The `show` command displays the reference for the child, `colorful`.

```
> list
celeste _default color colorful quartz nuttree puffin

> show colorful
printer
Reference type: onc_printers
Address type: onc_printers_bsdaddr
  length: 12
  data: 0x00 0x00 0x00 0x08 0x62 0x61 0x63 0x6b 0x63 ....blackc 0x61 0x74 at

> down colorful
Could not construct context handle: No Supported Address
> quit
%```
**Printer Programming Example**

Printer client and server software can take advantage of FNS to advertise and to browse the printers available with respect to organizations, sites, users and hosts. The APIs used by the server and the client are XFN APIs, thereby ensuring that the application will be portable across the different naming services used for storing printer bindings.

The programming example in this section shows how printer clients and servers obtain and store printer bindings. Users can then make use of the FNS commands, `fnlist` and `fnlookup`, to browse the printer context.

For example, use `fnlist` to look at the user `printer` context for `jsmith`:

```
% fnlist user/jsmith/service/printer
  celeste
  lp
  _default
  myprinter
```

Similarly, you can look at the organization’s printers:

```
% fnlist org/wiz.com/service/printer
  sales_printer
  mktg_printer
  eng_printer
```

Alternatively, you can type

```
% fnlist thisorgunit/service/printer
```

You can look at the printers at a specific site, for example, the printers in the MTV site:

```
% fnlist thisorgunit/site/MTV/service/printer
  b1_printer
  b2_printer
```
Client

The scenario for Code Example 12-2 is a user who would like to print to a printer named colorful in his organization's context, thisorgunit/service/printer/colorful. The example printer client illustrates how the bindings for a specific printer are obtained.

The variable `printer_binding` contains the reference (the binding information) of the named printer. Using the binding information, the printer client can connect to the server and send the printer request. Note that the `fn_ctx_lookup()` function can be replaced by `fn_ctx_list_name()` or `fn_ctx_list_bindings()` to list all the names and their bindings.

**Code Example 12-2  Printer Client**

```c
#include <stdio.h>
#include <xfn/xfn.h>
#include <string.h>
#include <stdlib.h>

/* Routine to obtain the address of a specific printer */

/* This routine takes the printer name and the address type as the input arguments and returns the address of the requested printer */

char *
get_address_of_printer(const char *printer_name,
                        const char *address_type)
{
    /* Variable list */
    FN_string_t *printer_name_string;
    FN_composite_name_t *printer_name_comp;
    FN_status_t *status;
    FN_ctx_t *initial_context;
    FN_ref_t *printer_ref;
    const FN_identifier_t *addr_id;
    const FN_ref_addr_t *address;
    char *addr_data; /* Return value */
    void *ip;
    size_t address_type_len, addr_len;

    /* Convert the printer name to a composite name */
    printer_name_string = fn_string_from_str((unsigned char *) printer_name);
```
printer_name_comp =
    fn_composite_name_from_string(printer_name_string);
fn_string_destroy(printer_name_string);

/* Get the initial context */
status = fn_status_create();
initial_context = fn_ctx_handle_from_initial(status);
/* Check status for any error messages */
if (!fn_status_is_success(status)){
    fprintf(stderr, "Unable to obtain the initial context\n");
    return (0);
}
/* Perform a lookup for the printer name */
printer_ref = fn_ctx_lookup(initial_context,
    printer_name_comp, status);
/* Check status for any error messages */
if (!fn_status_is_success(status)){
    fprintf(stderr, "Lookup failed on: %s\n",
        printer_name);
    return (0);
}
fn_ctx_handle_destroy(initial_context);
fn_composite_name_destroy(printer_name_comp);
address_type_len = strlen(address_type);

/* Obtain the requested address from the address type */
for (address = fn_ref_first(printer_ref, &ip);
    address != NULL;
    address = fn_ref_next(printer_ref, &ip)) { 
    addr_id = fn_ref_addr_type(address);
    if (addr_id->length == address_type_len &&
        strncmp(address_type, (char *)addr_id->contents,
            address_type_len) == 0)
        break;
}
if (address == NULL)
    return (0);

addr_len = fn_ref_addr_length(address);
addr_data = (char *)(malloc(addr_len + 1));
strcpy(addr_data, (char *)(fn_ref_addr_data(address)),
    addr_len);
addr_data[addr_len] = '\0';
Calling the Printer Client Function

The following code could be used to call the `get_address_of_printer()` routine shown above.

```c
char* printer_server;
printer_server = get_address_of_printer(
   "thisorgunit/service/printer/colorful",
   "onc_bsdaddr");
```

Server

Using the XFN APIs, print servers can advertise their services. Code Example 12-3 illustrates a host, `labpc`, that would like to advertise the binding for the color printer `colorful`. The FNS name for this printer is `thisorgunit/service/printer/colorful`.

The main tasks are to obtain the composite name for the printer name, to generate the binding (reference) for the printer, and to bind the name and references to the FNS namespace.

**Code Example 12-3  Printer Server**

```c
#include <stdio.h>
#include <xfn/xfn.h>
#include <string.h>

/* Routine to export the printer binding to the FNS name space */
/* This routine takes the printer name along with its reference type,  
   address type, and address. Returns the status. */
int
export_printer_to_fns(const char *printer_name,  
   const char *reference_type,  
   const char *address_type,  
   const char *address_data)
{
   /* Variable list */

   fn_ref_destroy(printer_ref);
   return (addr_data);
}
```
int return_status;
FN_string_t *printer_name_string;
FN_composite_name_t *printer_name_comp;
FN_identifier_t ref_id, addr_id;
FN_status_t *status;
FN_ref_t *printer_ref;
FN_ref_addr_t *address;
FN_ctx_t *initial_context;

/* Obtain the initial context */
status = fn_status_create();
initial_context = fn_ctx_handle_from_initial(status);
/* Check status for any error messages */
if ((return_status = fn_status_code(status)) != FN_SUCCESS) {
    fprintf(stderr, "Unable to obtain the initial context\n");
    return (return_status);
}

/* Construct the composite name for the printer name */
printer_name_string = fn_string_from_str((unsigned char *) printer_name);
printer_name_comp = fn_composite_name_from_string(printer_name_string);
fn_string_destroy(printer_name_string);

/* Construct the printer address */
addr_id.format = FN_ID_STRING;
addr_id.length = strlen(address_type);
addr_id.contents = (void *) address_type;
address = fn_ref_addr_create(&addr_id,
    strlen(address_data), (const void *) address_data);

/* Construct the printer reference */
ref_id.format = FN_ID_STRING;
ref_id.length = strlen(reference_type);
ref_id.contents = (void *) reference_type;
printer_ref = fn_ref_create(&ref_id);

/* Add the printer address to the printer reference */
fn_ref_append_addr(printer_ref, address);

/* Bind the reference to the context */
fn_ctx_bind(initial_context, printer_name_comp, printer_ref, 0, status);

/* Check the error status and return */
return_status = fn_status_code(status);
fn_composite_name_destroy(printer_name_comp);
fn_ref_addr_destroy(address);
fn_ref_destroy(printer_ref);
fn_status_destroy(status);
fn_ctx_handle_destroy(initial_context);
return (return_status);
}

Calling the Printer Server Function

The following code could be used to call the export_printer_to_fns routine shown above.

export_printer_to_fns(
    "thisorgunit/service/printer/colorful",
    "onc_printers",
    "onc_bsdaddr",
    "labpc");
This appendix provides supplemental information about XFN composite name syntax.

<table>
<thead>
<tr>
<th>Composite Name Encoding</th>
<th>page 185</th>
</tr>
</thead>
<tbody>
<tr>
<td>Backus-Naur Form (BNF)</td>
<td>page 186</td>
</tr>
<tr>
<td>Decomposing the Composite Name String</td>
<td>page 187</td>
</tr>
<tr>
<td>Composing the Composite Name String</td>
<td>page 189</td>
</tr>
</tbody>
</table>

**Composite Name Encoding**

All XFN implementations are required to support the ISO 646 portable representation (same encoding as ASCII) for XFN composite names. All other representations are optional.

All characters of the string form of an XFN composite name use a single encoding. There cannot be characters with different encodings in the same name string. This does not preclude component names of a composite name in its structural form from having different encodings. Code set mismatches that occur during the process of converting a composite name structure to its string form are resolved in an implementation-dependent way. Strings with code sets that are determined by the implementation to be compatible are converted without loss of information into a single representation, which is also determined by the implementation. When an implementation discovers that a composite name has components with incompatible code sets, it returns the error code FN_E__INCOMPATIBLE_CODE_SETS.
Backus-Naur Form (BNF)

The following defines the standard string form of XFN composite names in Backus-Naur Form (BNF). Note that all the characters of the string representation of one name must uniformly use the same encoding and locale information. The notations used are as follows:

Table 12-2 Backus-Naur Notation

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>::=</td>
<td>Is defined to be</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;text!&gt;</td>
<td>Nonterminal element</td>
</tr>
<tr>
<td>''</td>
<td>Literal expression</td>
</tr>
<tr>
<td>*</td>
<td>The preceding syntactic unit can appear 0 or more times.</td>
</tr>
<tr>
<td>+</td>
<td>The preceding syntactic unit can appear 1 or more times.</td>
</tr>
<tr>
<td>[]</td>
<td>The enclosed syntactic units are grouped as a single syntactic unit (can be nested).</td>
</tr>
</tbody>
</table>

The XFN composite name syntax in BNF is as follows.

Table 12-3 XFN Composite Name Syntax Using BNF

<table>
<thead>
<tr>
<th>XFN Composite Name</th>
<th>BNF Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>NULL ::=</td>
<td>// Empty set</td>
</tr>
<tr>
<td>&lt;PCS&gt; ::=</td>
<td>// Portable Character Set</td>
</tr>
<tr>
<td></td>
<td>The set consists of the glyphs:</td>
</tr>
<tr>
<td></td>
<td>// !&quot;#$%&amp;'()*+,-.0123456789:;&lt;=&gt;?</td>
</tr>
<tr>
<td></td>
<td>// @ABCDEFGHIJKLMNOPQRSTUVWXYZ[]^_`</td>
</tr>
<tr>
<td></td>
<td>// abcdefghijklmnopqrstuvwxyz{</td>
</tr>
<tr>
<td>&lt;CharSet&gt; ::=</td>
<td>&lt;PCS&gt;</td>
</tr>
<tr>
<td>&lt;EscapeChar&gt; ::=</td>
<td></td>
</tr>
<tr>
<td>&lt;ComponentSep&gt; ::=</td>
<td>/</td>
</tr>
<tr>
<td>&lt;Quote1&gt; ::=</td>
<td>&quot;</td>
</tr>
<tr>
<td>&lt;Quote2&gt; ::=</td>
<td>'</td>
</tr>
</tbody>
</table>
A

Table 12-3 XFN Composite Name Syntax Using BNF (Continued)

<table>
<thead>
<tr>
<th>XFN Composite Name</th>
<th>BNF Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;MetaChar&gt;</code> ::=</td>
<td>`&lt;EscapeChar&gt;</td>
</tr>
</tbody>
</table>
| `<SimpleChar>` ::=  | // any character from `<CharSet>` with `<MetaChar>`, `<Quote1>`, // and `<Quote2>` excluded. An `<EscapeChar> <MetaChar>`, or // `<EscapeChar> <Quote1>`, or `<EscapeChar> <Quote2>` is equivalent to // `<SimpleChar>`.
| `<Component>` ::=   | `<SimpleChar>`* |
|                    | | `<SimpleChar>`+ `{<Quote1> | `<Quote2>` | `<SimpleChar>`}`* |
|                    | | `<SimpleChar>`<CharSet>`<EscapeChar><Quote1>`*<Quote1>` |
|                    | // `<CharSet>` must not contain unescaped `<Quote1>` |
|                    | // (note that `<Quote2>` can appear unescaped) |
|                    | | `<Quote2>`<CharSet>`<EscapeChar><Quote2>`*<Quote2>` |
|                    | // `<CharSet>` must not contain unescaped `<Quote2>` |
|                    | // (note that `<Quote1>` can appear unescaped) |
| `<CompositeName>` ::= | NULL |
|                    | | `<Component>` `{<ComponentSep> <Component>}`* |

Decomposing the Composite Name String

The function `fn_composite_name_from_string()` returns an XFN composite name in its structural form, `FN_composite_name_t`, given the composite name’s string representation. The syntax rules used by `fn_composite_name_from_string()` are as follows.

An XFN composite name is decomposed into an ordered set of components `<Component>`. Each component represents a compound name, or a single atomic name of a compound name if the compound name’s syntax uses the XFN component separator (`/`) as a separator for its atomic parts and the compound name is not quoted.

The following are the rules for parsing a composite name.

1. Any `<ComponentSep>` character that is neither escaped nor enclosed in quoted strings is considered to be a component separator.

2. Any string enclosed by component separators is a component `<Component>`.
3. A composite name is parsed and decomposed into components from left to right:

a. The first component is the string preceding the first occurrence of a component separator.

b. Empty components are processed as follows:

i. A leading component separator (the composite name begins with a component separator) means a leading null component.

ii. A trailing component separator (the composite name ends with a component separator) means a trailing null component.

c. Two consecutive component separators mean a null component.

d. The name string that immediately follows the last component separator of the composite name is the final component.

4. A component string is evaluated from left to right and converted into its standard form according to the following rules:

a. A component string is considered to be quoted if it is enclosed in a pair of matching unescaped quote characters (either a <Quote1> or a <Quote2> pair). The quoted string must represent the full component; that is, a begin quote must immediately be preceded by a component separator or no character, and the end quote must immediately be followed by a component separator or no character.

b. If a component does not contain a valid begin quote (a <Quote1> or <Quote2> immediately preceded by either a component separator or no character), any occurrence of <Quote1> or <Quote2> within that component is treated just as any other <SimpleChar>.

c. An unmatched begin quote (missing or misplaced end quote) fails with an FN_E_ILLEGAL_NAME status.

d. Quotes are considered to be escaped in quoted strings if a matching quote character is preceded immediately by the unescaped <EscapeChar>.

e. Quoted components are resolved by eliminating the quote characters from the component name and substituting possibly escaped quotes by simple quote characters. <MetaChar>s and the nonmatching quote characters enclosed in quoted strings are treated just as any other <SimpleChar>. 
f. Any of the defined metacharacters (<ComponentSep> and <EscapeChar>) is considered to be escaped in an unquoted component name string if preceded immediately by the unescaped <EscapeChar> (for instance, the sequence <EscapeChar> <EscapeChar> <ComponentSep> denotes an escaped <EscapeChar> but an unescaped <ComponentSep>).

g. <Quote1> and <Quote2> are considered to be escaped in an unquoted component if and only if <EscapeChar> is preceded by a component separator (that is, sequences <ComponentSep> <EscapeChar> <Quote1> or <ComponentSep> <EscapeChar> <Quote2>). Other occurrences of <Quote1> and <Quote2> in an unquoted component are treated just as any other <SimpleChar>.

h. Any occurrence of escaped <MetaChar>, escaped <Quote1>, or escaped <Quote2> in unquoted components is substituted by the corresponding unescaped character.

i. No substitution is done for <EscapeChar> <SimpleChar>. <EscapeChar> <SimpleChar> simply maps to <EscapeChar> <SimpleChar>.

**Composing the Composite Name String**

The function fn_string_from_composite_name() returns the string representation of an XFN composite name given its structural form (FN_composite_name_t). The following are the rules used by fn_string_from_composite_name().

1. The components are added to the composite name string in left to right order (that is, rightmost is the tail).

2. Successive components are separated by the component separator (<ComponentSep>).

3. Empty components are handled in the following way:
   a. A leading empty component is represented by a leading <ComponentSep>.
   b. A trailing empty component is represented by a trailing <ComponentSep>.
   c. An empty component occurring within a composite name is represented by two consecutive <ComponentSep>.
4. A composite name denoting a single non-empty component does not contain any unescaped component separator.

5. Any occurrence of <ComponentSep> in a component is escaped by inserting <EscapeChar> immediately preceding <ComponentSep>.

6. If the first character of a component is either <Quote1> or <Quote2>, it will be escaped by inserting <EscapeChar> immediately preceding the quote.

7. Any occurrence of <EscapeChar> before <ComponentSep> in a component is escaped by inserting <EscapeChar> immediately preceding the <EscapeChar>.

8. Any occurrence of <EscapeChar> as the first character of a component with <Quote1> or <Quote2> as the second character in a component is escaped by inserting <EscapeChar> immediately preceding the <EscapeChar>. Subsequent <EscapeChar> occurring before any matching quote character is also escaped by inserting <EscapeChar> immediately preceding the <EscapeChar>.
DNS Text Record Format for XFN References

This appendix contains supplemental information about the use of DNS text (TXT) records in XFN references. The required procedures for federating DNS are contained in Chapter 7, “Federating NIS+ With Global Naming Systems.”

The Solaris environment conforms to the XFN specification for federating global naming systems within DNS. In order to federate a naming system under DNS, you will need to enter information into DNS TXT resource records. This information will then be used to construct an XFN reference for that subordinate naming system. This appendix describes the format of these DNS TXT records. For details on how to manipulate records in DNS in general, see DNS and BIND in a Nutshell by Paul Albitz and Crickett Liu (O’Reilly and Associates, 1992).

The reference type of an XFN reference is constructed from a TXT record that begins with the XFNREF tag. It has the following format:

```
TXT "XFNREF rformat reftype"
```

If spaces occur within the string appearing after TXT, such spaces must be escaped, or the entire string must be enclosed within double quotation marks. The three fields, XFNREF, rformat and reftype, are separated using space (spaces and tabs). rformat specifies format of the reference type identifier. It can be one of

- **STRING** – Maps to FN_ID_STRING format
- **OID** – Maps to FN_ID_ISO_OID_STRING format
The address information for an XFN reference is constructed using TXT records with tags prefixed with the XFN string. Multiple addresses may be specified for a single reference. Records with the same tag are grouped and passed to the handler for each group. Each handler generates zero or more addresses from its group of TXT records and appends the addresses to the reference. The XFNREF tag is special in that it is used only to construct the reference type and thus, it is excluded from the address-construction process.

The syntax of address TXT records is as follows:

```
XFN_address_type_tag   address_specific_data
```

The two fields, XFN_address_type_tag and address_specific_data, are separated using space (spaces and tabs). XFN_address_type_tag specifies the handler to be used for address_specific_data.

TXT records have a limitation of 2Kbytes of characters per record. If the address-specific data is too long to be stored in a single TXT record, multiple TXT records may be used, as shown:

```
TXT "XFN_address_type_tag address_specific_data1"
TXT "XFN_address_type_tag address_specific_data2"
```

When the tag-specific handler is called, both records are passed to it. The handler is responsible for determining the order in which these two lines need to be interpreted.
The order in which TXT records appear is not significant. If lines with different tags are present, lines with the same tag are grouped together before the tag-specific handler is called. In the following example, the handler for tag1 will be called with two text lines, and the handler for tag2 will be called with three text lines.

Here are some examples of TXT records that can be used for XFN references.

Example 1

```
TXT "XFNtag1 address_specific_data1"
TXT "XFNtag2 address_specific_data2"
TXT "XFNtag1 address_specific_data3"
TXT "XFNtag2 address_specific_data4"
TXT "XFNtag2 address_specific_data5"
```

Example 2

```
TXT "XFNREF STRING XFN_SERVICE"
TXT "XFNNISPLUS wiz.com. nis_master 129.144.40.23"
```
The following is an example of a DNS table with a subordinate naming system bound in it.

```
$ORIGIN test.sun.com
@ IN SOA foo root.eng.sun.com
    { 100 ;; Serial
        3600 ;; Refresh
        3600 ;; Retry
        3600 ;; Expire
        3600 ;; Minimum
    }
NS nshost
TXT "XFNREF STRING XFN_SERVICE"
TXT "XFNISPLUS wiz.com. nis_master 129.144.40.23"
nshost IN A 129.144.40.21
```
This appendix contains supplemental information about the use of X.500 attributes for XFN references. The required procedures for federating X.500 are contained in Chapter 7, “Federating NIS+ With Global Naming Systems.”

In order to permit an XFN reference to be stored as an attribute in X.500, the directory schema must be modified to support the object classes and attributes defined in this appendix. *Managing the X.500 Client Toolkit* includes information about modifying the X.500 directory schema.

### Object Classes

Two new object classes, XFN and XFN-supplement, are introduced to support XFN references. The XFN object class is not relevant in FNS since SunSoft’s X.500 directory product cannot support the introduction of new compound ASN.1 syntaxes. Instead, FNS uses the XFN-supplement object class.
The two new object classes are defined in ASN.1 as follows:

```asn1
xFN OBJECT-CLASS ::= {
  SUBCLASS OF { top }
  KIND auxiliary
  MAY CONTAIN { objectReferenceId |
                 objectReference |
                 nNSReferenceId |
                 nNSReference }
  ID id-oc-xFN
}

id-oc-xFN OBJECT IDENTIFIER ::= {
  iso(1) member-body(2) ansi(840) sun(113536)
  ds-oc-xFN(24)
}

xFNSupplement OBJECT-CLASS ::= {
  SUBCLASS OF { top }
  KIND auxiliary
  MAY CONTAIN { objectReferenceString |
                nNSReferenceString }
  ID id-oc-xFNSupplement
}

id-oc-xFNSupplement OBJECT IDENTIFIER ::= {
  iso(1) member-body(2) ansi(840) sun(113536)
  ds-oc-xFNSupplement(25)
}
```

The XFN-supplement object class is defined as an auxiliary object class so that it may be inherited by all X.500 object classes. It is defined with two optional attributes:

- `objectReferenceString` is used to hold an XFN reference to the object itself.
- `nNSReferenceString` is used to hold an XFN reference to a next naming system.
Both attributes are defined in ASN.1 as follows:

```asn1
objectReferenceString ATTRIBUTE ::= {
    WITH SYNTAX OCTET STRING
    EQUALITY MATCHING RULE octetStringMatch
    SINGLE VALUE TRUE
    ID { id-at-objectReferenceString }
}

id-at-objectReferenceString OBJECT IDENTIFIER ::= {
    iso(1) member-body(2) ansi(840) sun(113536)
    ds-at-objectReferenceString(30)
}

nNSReferenceString ATTRIBUTE ::= {
    WITH SYNTAX OCTET STRING
    EQUALITY MATCHING RULE octetStringMatch
    SINGLE VALUE TRUE
    ID { id-at-nNSReferenceString }
}

id-at-nNSReferenceString OBJECT IDENTIFIER ::= {
    iso(1) member-body(2) ansi(840) sun(113536)
    ds-at-nNSReferenceString(31)
}
```
Both `objectReferenceString` and `nNSReferenceString` store XFN references in a string form. Their octet string syntax is further constrained to conform to the following BNF definition:

```
<ref> ::= <id> '}{$' <ref-addr-set>
<ref-addr-set> ::= <ref-addr> | <ref-addr> '}{$' <ref-addr-set>
<ref-addr> ::= <id> '}{$' <addr-set>
<addr> ::= <hex-string>

{id} ::= 'id' '}{$' <string> | 'uuid' '}{$' <uuid-string> | 'oid' '}{$' <oid-string>

<string> ::= <char> | <char> <string>
<char> ::= <PCS> | '\'. <PCS>
<PCS> ::= // Portable Character Set:
  // "!"#'$%'()'*+,./0123456789:;<=>?@
  // @ABCDEFGHIJKLMNOPQRSTUVWXYZ[]_
  // 'abcdefghijklmnopqrstuvwxyz{|}~

<uuid-string> ::= <uuid-char> | <uuid-char> <uuid-string>
<uuid-char> ::= <hex-digit> | '-'

<oid-string> ::= <oid-char> | <oid-char> <oid-string>
<oid-char> ::= <digit> | '.'

<hex-string> ::= <hex-octet> | <hex-octet> <hex-string>
<hex-octet> ::= <hex-digit> <hex-digit>
<hex-digit> ::= <digit> | 'a' | 'b' | 'c' | 'd' | 'e' | 'f' | 'A' | 'B' | 'C' | 'D' | 'E' | 'F'
<digit> ::= '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'
```

The following example is a string form XFN reference:

```
id$onc_fn_enterprise$id$onc_fn_nisplus_root$00000077697a2e636fd2e2062696762696700
```
The example uses an XFN reference of type `onc_fn_enterprise`. It contains the address type `onc_fn_nisplus_root` and a single address value. The address value is an XDR-encoded string, comprising the domain name, `wiz.com`, followed by the host name, `bigbig`.

An XFN reference may be added to an X.500 entry by using the FNS command `fnattr`, as in this example:

```
# fnattr -a .../c=us/o=wiz object-class top organization xfn-supplement
```

creates a new entry called `c=us/o=wiz` and adds an object class attribute with the values `top`, `organization`, and `XFN-supplement`.

The FNS command `fnbind` binds the NIS+ reference to the named entry and links X.500 to the root of the NIS+ namespace. (Note the use of a trailing slash in the name argument to `fnbind`.)

```
# fnbind -r .../c=us/o=wiz/ onc_fn_enterprise onc_fn_nisplus_root "wiz.com. bigbig"
```
**Glossary**

**application-level name service**
Application-level name services are incorporated in applications offering services such as files, mail, and printing. Application-level name services are bound below enterprise-level name services. The enterprise-level name services provide contexts in which contexts of application-level name services can be bound.

**atomic name**
An indivisible component of a name as defined by the naming convention.

**attribute**
Each named object is associated with a set of zero or more attributes. Each attribute in the set has a unique attribute identifier, an attribute syntax, and a set of zero or more distinct attribute values.

**binding**
The association of an atomic name with an object reference. For simplicity, an object reference and the object it refers to are used interchangeably in this guide.

**BNF**
Backus-Naur Form.

**composite name**
A name that spans multiple naming systems. It consists of an ordered list of zero or more components. Each component is a name from the namespace of a single naming system. Composite name resolution is the process of resolving a name that spans multiple naming systems.
compound name
A sequence of atomic names composed according to the naming convention of a naming system.

context
An object whose state is a set of bindings with distinct atomic names. Every context has an associated naming convention. A context provides a lookup (resolution) operation, which returns the reference, and may provide operations such as binding names, unbinding names, and listing bound names.

DNS
Domain Name System. A system that provides the naming policy and mechanisms for mapping domain and machine names to addresses on the Internet.

enterprise-level name service
A name service that names objects within an enterprise. The types of objects named are organizational units, sites, users, hosts, and files. Enterprise-level name services are bound below global name services. Global name services provide contexts in which the root contexts of enterprise-level naming systems can be bound.

enterprise root
The root context of an enterprise. A context for naming objects found at the root of the enterprise namespace.

federated naming service
The service offered by a federated naming system.

federated naming system
An aggregation of autonomous naming systems that cooperate to support name resolution of composite names through a standard interface. Each member of a federation has autonomy in its choice of operations other than name resolution.

federated namespace
The set of all possible names generated according to the policies that govern the relationships among member naming systems and their respective namespaces.

generic context
A context for binding names used in applications.
<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>global context</td>
<td>A context for naming objects that have global names (currently, DNS and X.500 are the only global naming systems specified by XFN).</td>
</tr>
<tr>
<td>global name service</td>
<td>A name service that has worldwide scope, such as Internet DNS and X.500. The types of entities named at this global level are typically countries, states, provinces, cities, companies, universities, institutions, and government departments and ministries. Each of these entities can be an enterprise.</td>
</tr>
<tr>
<td>host context</td>
<td>A context for naming objects related to a computer.</td>
</tr>
<tr>
<td>implicit naming system pointer</td>
<td>An unnamed reference that points to a context in another naming system.</td>
</tr>
<tr>
<td>initial context</td>
<td>Every XFN name is interpreted relative to some context, and every XFN naming operation is performed on a context object. The XFN interface provides a function that allows the client to obtain an initial context object that provides a starting point for resolution of composite names.</td>
</tr>
<tr>
<td>junction</td>
<td>A name in one namespace bound to a context in the next naming system.</td>
</tr>
<tr>
<td>naming convention</td>
<td>Every name is generated by a set of syntactic rules called a naming convention.</td>
</tr>
<tr>
<td>namespace</td>
<td>The set of all names in a naming system.</td>
</tr>
<tr>
<td>namespace identifier</td>
<td>A special atomic name used to refer to the root of a namespace.</td>
</tr>
<tr>
<td>name service</td>
<td>The service offered by a naming system. It is accessed through its interface.</td>
</tr>
<tr>
<td>naming system</td>
<td>A connected set of contexts of the same type (having the same naming convention) and providing the same set of operations with identical semantics. In the UNIX operating system, for example, the set of directories in a given file system (and the naming operations on directories) constitutes a naming system.</td>
</tr>
</tbody>
</table>
next naming system pointer (NNSP)
Reference to a context in which composite names from subordinate naming systems are resolved.

organizational units
An enterprise is organized into organizational units such as centers, laboratories, departments, divisions, and so on. An organizational unit is a subunit of an enterprise.

organizational unit context
A context for naming objects related to an organizational unit within an enterprise.

parent context
A context in which this context and its siblings are bound.

reference
The thing bound to a name. It contains addresses identifying the communication endpoints of the object.

root context
A context for naming the objects found in the root of the namespace.

service context
A context for naming objects that provide services.

site context
A context for naming objects related to a physical site.

strong separation
The case where the XFN context treats the XFN component separator as the naming system boundary.

subcontext
A context bound within another context.

user context
A context for naming objects related to a human user.

weak separation
The case where the XFN context does not treat the XFN component separator as the naming system boundary.
<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>XFN link</td>
<td>A special form of reference that has a composite name as an address. Like any other type of reference, an XFN link is bound to an atomic name in a context.</td>
</tr>
<tr>
<td>X.500</td>
<td>A global-level directory service defined by an Open Systems Interconnection (OSI) standard.</td>
</tr>
</tbody>
</table>
Index

Symbols

" (quotation marks)
BNF notation, 186
XFN composite name syntax, 159
XFN standard syntax model, 157
* in BNF notation, 186
+ in BNF notation, 186
. (dots)
... namespace identifier, 61, 62
/... namespace identifier, 62
trailing, in organization names, 31, 38
/ (slashes)
/... namespace identifier, 62
double slashes in organization names, 45, 99
FNS component separator, 42
in service names, 39
trailing, in organization names, 100
XFN component separator, 157, 159 to 161
::= in BNF notation, 186
\ as XFN component escape character, 159
_ (underscore)
in namespace identifiers, 41, 56
in XFN terminology, 42
| (pipe) in BNF notation, 186
’ (single quote) in XFN composite name syntax, 159

Numerics

0 value, 135

A
abstract data types, 134
access control
attribute no permission message, 101
changing, 99
checking, 98 to 99
no permission message, 103, 105 to 106
addresses
mail destinations, 33
multiple, 11
NIS+ root reference, 111 to 112
no supported address message, 103
properties, 12
references, 11 to 12
XFN interface parameters, 153 to 154
administraton
federating NIS+ with global naming systems, 111 to ??
obtaining NIS+ root reference, 111 to 112
under DNS, 112 to ??
under X.500, 114 to ??
file system namespace, 117 to 126
automounter, 119 to 120
file context administration, 126
file context creation, 120 to 126
NFS file servers, 118 to 119
overview, 117 to 120
FNS on NIS+, 69 to 109
access control checking, 98 to 99
browsing FNS structures using NIS+ commands, 97
error messages, 100 to 104
FNS attribute management, 92 to 94
FNS context creation, 73 to 83
FNS context management, 83 to 92, 141 to 142
maintaining consistency between NIS+ and FNS, 94 to 96
mapping FNS contexts to NIS+ objects, 96
replicating FNS service, 72
resource requirements, 70
setting up FNS namespace, 71 to 72
setting up NIS+ service for FNS, 70 to 71
troubleshooting, 104 to 109
printer namespace, 127 to 130
overview, 127
using files, 128
using NIS, 129
using NIS+, 129 to 130
alias host names, 77
all hosts context
creating, 74, 76
troubleshooting creation, 107
all users context
creating, 74, 77
troubleshooting creation, 107
API usage model, 22
application programming
namespace browser example, 168 to 177
code, 168 to 175
commands, 175
diagram, 168
sample output, 176 to 177
printer example, 178 to 183
client, 179 to 181
server, 181 to 183
XFN composite names, 159 to 166
naming system boundaries and component separators, 161 to 163
resolution, 163 to 166
syntax, 159 to 161
XFN interfaces, 133 to 158
See also client programming interfaces
abstract data types, 134
base attribute interface, 143 to 149
base context interface, 135 to 143
conventions, 134
memory management policies, 135
overview, 133 to 135
parameters, 153 to 155
parsing compound names, 156 to 158
status codes, 150 to 153
status objects, 137, 145, 150
usage, 134
applications
API usage model, 22
DeskSet tools, 32 to 35
FNS implementation, 10
FNS interaction, 19 to 21
name services, 26, 27
architectural model, 11 to 16
attributes, 13
composite names, 15 to 16
compound names, 13 to 14
contexts, 12
initial context, 16
references, 11 to 12
XFN links, 15
ASCII string XFN identifier format, 154
asterisk (*) in BNF notation, 186
atomic names
See also namespace identifiers
in compound names, 13
in contexts, 12
global level, 62
initial context bindings for global
naming, 62
namespace identifiers in the
enterprise, 41 to 42
attribute no permission
message, 101
attribute operations
See also base attribute interface;
context operations
attribute-modification
operations, 146
get attribute, 145 to 146
get attribute identifiers, 148
get attribute values, 147
get multiple attributes, 148 to 149
modify attribute, 146
modify multiple attributes, 149
multiple-attribute operations, 147 to 149
relationship to naming
operations, 144 to 145
single-attribute operations, 145 to 147
status objects, 145
XFN attribute model, 143 to 144
attribute value required
message, 101
attribute-modification lists, 155
attributes
adding attributes or values, 92, 93,
146
base attribute interface, 143 to 149
attribute-modification
operations, 146
multiple-attribute
operations, 147 to 149
relationship to naming
operations, 144 to 145
single-attribute operations, 145
to 147
status objects, 145
supporting interfaces, 133 to 134
XFN attribute model, 143 to 144
deleting attributes or values, 93
described, 13
error messages, 101, 102, 103, 104
getting, 145 to 146
identifiers, 148
multiple attributes, 148 to 149
syntax attributes of context, 143
values, 147
listing, 93
managing and examining, 92 to 94
modifying values, 94
replacing, 93
sets, 155
syntax attributes, 156, 158
getting, 143
X.500 attribute syntax for XFN
references, 195 to 199
XFN interface parameters, 155
XFN model, 143 to 144
authentication failure
message, 101
automounter, 119 to 120
multiple locations, 124
variable substitution, 125
backslash (\) as XFN component escape character, 159
Backus-Naur Form (BNF), 186 to 187
bad reference message, 101
base attribute interface, 143 to 149
See also client programming interfaces
abstract data types, 134
attribute-modification operations, 146
conventions, 134
memory management policies, 135
multiple-attribute operations, 147 to 149
parameters, 153 to 155
attribute modification lists, 155
attribute sets, 155
attributes and attribute values, 155
composite names, 153
identifiers, 154
references and addresses, 153 to 154
strings, 155
parsing compound names, 156 to 158
syntax attributes, 156, 158
XFN standard syntax model, 156 to 157
requirements for operations, 136 to 137
status objects, 137
supporting interfaces, 133 to 134
updating bindings, 140 to 141
usage, 134
begin quote (") in XFN standard syntax model, 157
See also double quotes
bind/lookup model, 22
bindings
access control changing, 99
checking, 98 to 99
adding, 140
calendar service example, 33
composite names to references, 89 to 91, 108
displaying, 83 to 84
initial context bindings for enterprise naming, 54 to 60
table, 55
example, 54
host-related bindings, 58 to 59
“shorthand” bindings, 59 to 60
user-related bindings, 56 to 57
initial context bindings for global naming, 62
listing names and bindings in contexts, 139
printers, 127
removing composite names, 91, 108
terminal atomic name, 140 to 141
renaming, 91, 141
updating, 140 to 141
BNF (Backus-Naur Form), 186 to 187
boundaries (naming system) and component separators, 161 to 163
strong separation, 161 to 162
weak separation, 162 to 163
browsing
See also displaying; listing; lookup operations FNS structures using NIS+ commands, 97
namespace browser programming example, 168 to 177
code, 168 to 175
commands, 175
diagram, 168
sample output, 176 to 177

C
Calendar Manager
calendar service example, 33 to 35
FNS policies, 32, 33 to 35
“Cannot obtain initial context” message, 104
canonical namespace identifiers, 42, 56
checking access control, 98 to 99
client programming interfaces, 133 to 158
See also application programming abstract data types, 134
base attribute interface, 143 to 149
attribute-modification operations, 146
multiple-attribute operations, 147 to 149
relationship to naming operations, 144 to 145
single-attribute operations, 145 to 147
status objects, 145
XFN attribute model, 143 to 144
base context interface, 135 to 143
context handles, 137
lookup and list contexts, 138 to 140
managing contexts, 141 to 142
names in context operations, 136
other context operations, 142 to 143
requirements for operations, 136 to 137
status objects, 137
updating bindings, 140 to 141
conventions, 134
memory management policies, 135
overview, 133 to 135
parameters, 153 to 155
attribute modification lists, 155
attribute sets, 155
attributes and attribute values, 155
composite names, 153
identifiers, 154
references and addresses, 153 to 154
strings, 155
parsing compound names, 156 to 158
syntax attributes, 156, 158
XFN standard syntax model, 156 to 157
status codes, 150 to 153
status objects, 150
base attribute interface, 145
base context interface, 137
supporting interfaces, 133 to 134
usage, 134
cm service, 33 to 35
code sets incompatible, 102
codes
attribute-modification operation, 146
link status, 150
status, 150 to 153
commands
See also attribute operations; context operations; specific commands, functions, and operations
browsing FNS structures using NIS+ commands, 97
fnbrowse program, 175
FNS context management, 83 to 92
XFN interface function names, 134
communication failure message, 101
component separator (/)
  FNS components, 42
  naming system boundaries and, 161 to 163
  strong separation, 161 to 162
  weak separation, 162 to 163
XFN composite name syntax, 159 to 161
XFN standard syntax model, 157
composing XFN composite name strings, 189 to 190
composite names
  applications’ use of FNS, 10
  binding to references, 89 to 91, 108
  defined, 4, 15
  destroying named objects, 92, 109
  displaying bindings, 83 to 84
  examples
    calendar service, 33 to 35
    hosts, 30
    illustration, 16
    organizations, 29
    sites, 30
    user, 29
host naming systems, 30
need for uniform policy, 6 to 7
organization naming systems, 29
parsing XFN composite names, 187 to 189
removing from namespace, 91 to 92, 108 to 109
resolution, 163 to 166
  coexistence of explicit and implicit NNSPs, 165
  explicit NNSPs, 163 to 164
  implicit NNSPs, 164 to 165
  XFN links, 165
site naming systems, 30
user naming systems, 29
XFN composite names, 159 to 166
  naming system boundaries and component separators, 161 to 163
  resolution, 163 to 166
  syntax, 159 to 161, 185 to 190
XFN context implementation, 136
XFN interface parameters, 153
XFN syntax, 159 to 161, 185 to 190
  Backus-Naur Form (BNF), 186 to 187
  composing the composite name string, 189 to 190
  decomposing the composite name string, 187 to 189
  encoding, 185
  string and structural forms, 160 to 161
compound names, 13 to 14
  described, 13
  hierarchical naming system examples, 13 to 14
  parsing, 156 to 158
  syntax attributes, 156, 158
  XFN standard syntax model, 156 to 157
configuration error message, 101
cost parameters, 135
cost pointers, 135
constants, XFN interface conventions, 134
context not empty message, 102
context operations
See also attribute operations; base context interface
bind, 140
construct context handle from reference, 137
construct handle to initial context, 137
context handles, 137
create subcontext, 141 to 142
destroy context handle, 143
destroy subcontext, 142
get reference to context, 142
get syntax attributes of context, 143
list bindings, 139
list names, 138 to 139
lookup, 138
lookup link, 139 to 140
managing contexts, 141 to 142
names in, 136
rename, 141
requirements, 136 to 137
status objects, 137
unbind, 140 to 141
updating bindings, 140 to 141
context shared object modules, 20
contexts
See also bindings; initial context; parent contexts; subordinate contexts
base context interface, 135 to 143
circuit handles, 137
lookup and list contexts, 138 to 140
managing contexts, 141 to 142
names in context operations, 136
other context operations, 142 to 143
requirements for operations, 136 to 137
status objects, 137
supporting interfaces, 133 to 134
updating bindings, 140 to 141
changing ownership, 99
checking naming inconsistencies, 95 to 96
context not empty message, 102
creating individually
all hosts context, 74, 76, 107
all users context, 74, 77, 107
debugging, 74
file context, 82
fncreate command
overview, 73 to 74
generic context, 74, 80 to 81
namespace identifier context, 74, 82
organization context, 75
printer context, 80
service context, 79 to 80
single host context, 76, 107
single user context, 78 to 79, 107
site context, 81
creating subcontexts, 141 to 142
defined, 12
destroying
handles, 143
subcontexts, 142
displaying the binding, 83 to 84
enterprise root, 45 to 46
files, 52
administering, 126
creating, 120 to 126
getting
handles, 137
references, 142
syntax attributes, 143
hosts, 50 to 51
initial context
bindings for enterprise naming, 54 to 60
calendar service example, 34
described, 16
listing contents, 85 to 88
managing and examining, 83 to 92, 141 to 142
mapping to NIS+ objects, 96
not a context message, 104
organizational units, 46 to 47
principles, 7 to 8
printers, 53, 128 to 130
services, 51 to 52
sites, 47 to 49
syntax-related attributes, 156
tree structure, 13 to 14
troubleshooting
cannot create host- or user-related contexts, 107
cannot remove context, 107
checking naming inconsistencies, 95 to 96
debugging creation, 74
users, 49
XFN contexts, 12
continue operation using status message, 102
ctx_dir directory
checkpointing, 72
creating, 71
replicating, 72
verifying creation, 71
delimiting, See removing
DeskSet tools, 32 to 35
destroying
See also removing
context handles, 143
named objects, 92
context not removed, 107
“operation failed” not returned, 109
subcontexts, 142
DIB (directory information base), 63 to 64
directories
automounter, 119 to 120
NIS+ master server, 71
directory information base (DIB), 63 to 64
disk-space requirements for FNS on NIS+, 70
displaying
See also listing; lookup operations
access control, 98 to 99
bindings, 83 to 84
browsing FNS structures using NIS+ commands, 97
DNS table, editing, 112 to ??
DNS, See Domain Name System (DNS)
Domain Name System (DNS)
as nonterminal name system, 63
described, 9
federating, 62 to 63
federating NIS+ under, 112 to ??
FNS implementation, 9
hierarchical naming system, 13
name resolution, 62
text record format for XFN references, 191 to 194
domains (NIS+) and FNS organizational units, 30 to 31
dots (.)
... namespace identifier, 61, 62
/... namespace identifier, 62
trailing, in organization names, 31, 38
double quotes
BNF notation, 186
XFN composite name syntax, 159
XFN standard syntax model, 157
E
editing the DNS table, 112 to ??
encoding for XFN composite names, 185
end quote (”) in XFN standard syntax model, 157
See also double quotes
enterprise level of service, 26, 27
enterprise namespace policies, 37 to 60
arrangement of objects, 28
file namespace, 40
host namespace, 39
illustrated, 27
initial context bindings, 54 to 60
example, 54
host-related bindings, 58 to 59
“shorthand” bindings, 59 to 60
table, 55
user-related bindings, 56 to 57
namespace identifiers, 41 to 42
namespace structure, 42 to 54
enterprise root, 45 to 46
example, 44
files, 52
hosts, 50 to 51
organizational units, 46 to 47
printers, 53
services, 51 to 52
sites, 47 to 49
users, 49
organizational unit namespace, 38
printer namespace, 40
service namespace, 39 to 40
site namespace, 38
table of policies, 43
user namespace, 39
enterprise root context, 45 to 46
enumeration handle invalid, 102
erasing, See removing
error message, 102
error messages, 100 to 104
See also troubleshooting
attribute no permission, 101
attribute value required, 101
authentication failure, 101
bad reference, 101
“Cannot obtain initial context”, 104
communication failure, 101
configuration error, 101
context not empty, 102
continue operation using
status values, 102
error, 102
illegal name, 102
incompatible code sets, 102
insufficient resources, 102
invalid attribute
identifier, 102
invalid attribute value, 102
invalid enumeration
handle, 102
invalid syntax attributes, 103
link error, 103
link loop limit reached, 103
malformed link, 103
name in use, 103, 108
name not found, 103
no permission, 103, 105 to 106
no such attribute, 103
no supported address, 103
not a context, 104
operation not supported, 104
overview, 100
partial result returned, 104
status codes, 151 to 153
success, 104
syntax not supported, 104
too many attribute values, 104
unavailable, 104
examining, See displaying; listing; lookup operations
explicit NNSPs, 163 to 164
See also next naming system pointers (NNSPs)
exporting the FNS interface, 10

F
-f, fncreate option
all hosts context, 76
all users context, 78
overview, 74
federated enterprise namespace policies,
See enterprise namespace policies
federated global namespace policies, See
global namespace policies
Federated Naming Service
API usage model, 22
application view, 19 to 21
architectural model, 11 to 16
browsing FNS structures using NIS+ commands, 97
described, xvii, 4
error messages, 100 to 104
file system view, 18
need for, 5 to 8
coherence in naming, 7 to 8
name composition uniformity, 6 to 7
naming interface uniformity, 5 to 6
principles for policies, 7 to 8
registry of service names, 40
setting up
FNS namespace setup, 71 to 72
NIS+ service setup, 70 to 71
replicating FNS service, 72
resource requirements, 70
Solaris environment, 8 to 10
user’s view, 17 to 18
XFN compliance, xvii, 5
XFN vs., 5
federating
DNS, 62 to 63
NIS+ with global naming systems, 111 to ??
getting NIS+ root reference, 111 to 112
under DNS, 112 to ??
under X.500, 114 to ??
X.500, 63 to 65
File Manager, 32
files and file systems
contexts, 52
creating, 82
ownership, 82
enterprise namespace, 40
enterprise namespace policies, 43
as enterprise policy entities, 28
FNS interaction, 18
FNS policies, 32
FNS-based naming, 9
input file for fncreate_fs(1M)
alternate format, 125
creating, 121 to 123
namespace administration, 117 to 126
automounter, 119 to 120
file context administration, 126
file context creation, 120 to 126
NFS file servers, 118 to 119
overview, 117 to 120
namespace identifier, 41
NFS file servers, 118 to 119
printer context administration, 128
FN_prefix, 134
fn_prefix, 134
fn_attr_get() function, 145 to 146, 147
fn_attr_get_ids() function, 148
fn_attr_get_values() function, 146, 147
fn_attr_modify() function, 146
fn_attr_multi_modify() function, 149
fn_attr_multiget() function, 148
FN_ATTRIB_ADD operation code, 146
FN_ATTRIB_ADD_EXCLUSIVE operation code, 146
FN_ATTRIB_ADD_VALUES operation code, 146
FN_ATTRIB_REMOVE operation code, 146
FN_ATTRIB_REMOVE_VALUES operation code, 146
fn_composite_name_from_string() function, 187 to 189
fn_ctx_bind() function, 140
fn_ctx_bindinglist_destroy() function, 139
fn_ctx_bindinglist_next() function, 139
fn_ctx_create_subcontext() function, 141 to 142
fn_ctx_destroy_subcontext() function, 142
fn_ctx_get_ref() function, 142
fn_ctx_get_syntax_attrs() function, 143
fn_ctx_handle_destroy() function, 143
fn_ctx_handle_from_initial() function
getting context handles, 137
getting initial context object, 54
host-related bindings, 58 to 59
user-related bindings, 56 to 57
fn_ctx_handle_from_ref() function, 137
fn_ctx_list_names() function, 138
fn_ctx_listbindings() function, 139
fn_ctx_lookup() function
support required, 136
using, 138
fn_ctx_lookup_link() function, 139 to 140
fn_ctx_namelist_destroy() function, 138
fn_ctx_namelist_next() function, 138
fn_ctx_rename() function, 141
fn_ctx_unbind() function, 140 to 141
FN_E_ATTR_NO_PERMISSION status code, 151
FN_E_ATTR_VALUE_REQUIRED status code, 151
FN_E_AUTHENTICATION_FAILURE status code, 151
FN_E_COMMUNICATION_FAILURE status code, 151
FN_E_CONFIGURATION_ERROR status code, 151
FN_E_CONTINUE status code, 151
FN_E_CTX_NO_PERMISSION status code, 151
FN_E_CTX_NOT_EMPTY status code, 151
FN_E_CTX_UNAVAILABLE status code, 151

FN_E_ILLEGAL_NAME status code, 151, 157
FN_E_INCOMPATIBLE_CODE_SETS status code, 151, 157
FN_E_INSUFFICIENT_RESOURCES status code, 151
FN_E_INVALID_ATTR_VALUE status code, 152
FN_E_INVALID_ENUM_HANDLE status code, 152
FN_E_INVALID_SYNTAX_ATTRS status code, 152
FN_E_LINK_ERROR status code, 150, 152
FN_E_LINK_LOOP_LIMIT status code, 152
FN_E_MALFORMED_LINK status code, 152
FN_E_MALFORMED_REFERENCE status code, 152
FN_E_NAME_IN_USE status code, 152
FN_E_NAME_NOT_FOUND status code, 152
FN_E_NO_SUCH_ATTRIBUTE status code, 152
FN_E_NO_SUPPORTED_ADDRESS status code, 152
FN_E_NOT_A_CLIENT status code, 152
FN_E_OPERATION_NOT_SUPPORTED status code, 136, 152
FN_E_PARTIAL_RESULT status code, 152
FN_E_SYNTAX_NOT_SUPPORTED status code, 153
FN_E_TOO_MANY_ATTR_VALUES status code, 153
FN_E_UNSPECIFIED_ERROR status code, 153
FN_ID_DCE_UUID XFN identifier format, 154
FN_ID_ISO_OID_BER XFN identifier format, 154
FN_ID_ISO_OID_STRING XFN identifier format, 154
FN_ID_STRING XFN identifier format, 154
fn_multigetlist_destroy() function, 148 to 149
fn_multigetlist_next() function, 148 to 149
FN_status_t parameter, 137
fn_std_syntax_ava_separator XFN syntax attribute, 158
fn_std_syntax_begin_quote XFN syntax attribute, 158
fn_std_syntax_case_insensitive XFN syntax attribute, 158
fn_std_syntax_code_sets XFN syntax attribute, 158
fn_std_syntax_end_quote XFN syntax attribute, 158
fn_std_syntax_escape XFN syntax attribute, 158
fn_std_syntax_local_info XFN syntax attribute, 158
fn_std_syntax_separator XFN syntax attribute, 158
fn_std_syntax_typeval_separator XFN syntax attribute, 158
fn_string_from_composite_name() function, 189 to 190
FN_SUCCESS status code, 151
fn_syntax_direction XFN syntax attribute, 158
fn_syntax_type XFN syntax attribute, 158
fn_valuelist_destroy() function, 147
fn_valuelist_next() function, 147
fnbind command
binding composite names to references, 89 to 90
name in use message with -s option, 108
options, 89
syntax, 83
fnbind -r command
binding composite names to references, 90 to 91
options, 89
syntax, 83
fnbrowse program example, 168 to 177
code, 168 to 175
commands, 175
diagram, 168
sample output, 176 to 177
fncheck command, 95 to 96
fncreate command
all hosts context creation, 74, 76, 107
all users context creation, 74, 77, 107
debugging context creation, 74
file context creation, 82
FNS namespace setup, 71 to 72
generic context creation, 74, 80 to 81
name in use message with -s option, 108
namespace identifier context creation, 74, 82
NIS_GROUP environment variable setting, 70
options, 74
organization context creation, 75
overview, 73 to 74
printer context creation, 80
service context, 80
service context creation, 79
single host context creation, 76, 107
single user context creation, 78 to 79, 107
site context creation, 81
syntax, 73
troubleshooting host- or user-related context creation, 107
fncreate_fs(1M) command
command-line input, 123 to 126
input file
 alternate format, 125
 creating, 121 to 123
 multiple locations, 124
 options, 121
 overview, 120 to 121
 syntax, 120
 variable substitution, 125

fndestroy command
 context not removed, 107
 destroying named objects, 92
 “operation failed” not returned, 109
 syntax, 83

fnlist command
 doesn’t list suborganizations, 106
 initial context contains nothing, 105
 listing context contents, 85 to 88
 options, 85
 syntax, 83

fnlookup command
 displaying bindings, 83 to 84
 options, 83
 syntax, 83

fnrename command
 renaming existing bindings, 91
 syntax, 83
 FNS, See Federated Naming Service
 fns_hosts table, 97
 fns_rserver creation, 72

fnunbind command
 name in use message, 108
 “operation failed” not returned, 109
 removing composite names, 91
 syntax, 83
 Font> message, 102

fs context type, 82
 See also files and file systems

fs or _fs namespace identifier
 See also files and file systems
 FNS policy, 43
 resolution, 41

functions
 See also specific functions
 XFN interface conventions, 134

G
generic context creation, 74, 80 to 81
generic context type, 80 to 81

going
 attribute identifiers, 148
 attribute values, 147
 attributes, 145 to 146
 context handles, 137
 multiple attributes, 148 to 149
 reference to context, 142
 syntax attributes of context, 143

global level of service, 26, 27

global namespace policies, 61 to 65
 federating DNS, 62 to 63
 federating X.500, 63 to 65
 global namespace, 61
 illustrated, 27
 initial context bindings, 62
 table of policies, 61

H
handles
 context handles
 destroying, 143
 getting, 137
 overview, 135

hard-disk space requirements for FNS on NIS+, 70

hierarchical naming system
 compound name examples, 13 to 14
 enterprise namespace structure, 44 to 45

di rectory context type, 76

direct or _host namespace identifier
 FNS policy, 43
 initial context binding, 55
 resolution, 41
 “shorthand” binding, 59

hostname context type, 39, 76
host-related bindings, 58 to 59
hosts
  aliases, 77
  as enterprise policy entities, 28
  bindings for enterprise naming, 58 to 59
  composite name examples, 30
  context creation
    all hosts, 74, 76
    single host, 76
    troubleshooting, 107
  context ownership, 76, 77
  contexts, 50 to 51
  enterprise namespace, 39
  enterprise namespace policies, 43
  namespace identifier, 41
hosts.org_dir table, 32

I
identifiers
  namespace
    canonical vs. Solaris
      customized, 42, 56
    context creation, 74, 82
    enterprise level, 41 to 42, 55
    global level, 61
    XFN interface parameters, 154
  illegal name message, 102
  implicit NNSPs, 164 to 165
    See also next naming system pointers (NNSPs)
  incompatible code sets
    message, 102
  inconsistencies
    checking context naming inconsistencies, 95 to 96
  initial context
    See also contexts
    bindings for enterprise naming, 54 to 60
      example, 54
      host-related bindings, 58 to 59
      “shorthand” bindings, 59 to 60
    table, 55
  user-related bindings, 56 to 57
  bindings for global naming, 62
  calendar service example, 34
  cannot obtain, 104
  contains nothing, 105
  described, 16
  handle construction operation, 137
  input file for fncreate_fs(1M)
    alternate format, 125
    creating, 121 to 123
  insufficient resources
    message, 102
  interfaces for programming, See client programming interfaces
  Internet DNS, See domain name system (DNS)
  invalid attribute identifier
    message, 102
  invalid attribute value
    message, 102
  invalid enumeration handle, 102
  invalid syntax attributes
    message, 103
  ISO OID XFN identifier formats, 154

J
junctions, 163 to 164
  See also next naming system pointers (NNSPs)

L
-L
  fnbind option, 89, 90
  fnlookup option, 83
-1
  fnattr option, 93
  fnlist option, 85, 87 to 88
  link error message, 103
  link loop limit reached message, 103
links (XFN)
  composite name resolution, 165
  creating and binding, 89, 90
  described, 15
  displaying binding, 83
  error messages, 103
  lookup operation, 139 to 140
  status object information, 150
  XFN header file, 134
  XFN library, 134
listing
  See also displaying; listing
  attributes and their values, 93
  browsing FNS structures using NIS+ commands, 97
  context contents, 85 to 88
  context naming inconsistencies, 95 to 96
  DNS domain contents, 62
  fns_hosts table contents, 97
  initial context contains nothing, 105
  names and bindings in contexts, 139
  names bound in contexts, 138 to 139
  namespace browser programming example, 168 to 177
  code, 168 to 175
  commands, 175
  diagram, 168
  sample output, 176 to 177
  NIS+ objects used by FNS, 97
  printing references used for binding, 89, 90
  suborganizations not listed, 106
lookup model, 22
lookup operations
  See also displaying; listing
calendar service example, 33
  contexts, 138
  XFN links, 139 to 140
managing, See administration
mapping FNS contexts to NIS+ objects, 96
memory management policies for client interfaces, 135
messages, See error messages
modules, context shared object, 20
mounting directories, 119 to 120
multiple addresses, 11
multiple attributes
  getting, 148 to 149
  getting identifiers, 148
  modifying, 149
myens or _myens namespace identifier
  initial context binding, 55
  user-related binding, 57
myorgunit or _myorgunit namespace identifier
  initial context binding, 55
  user-related binding, 57
myself or _myself namespace identifier
  initial context binding, 55
  user-related binding, 56
N
name in use message, 103
  with fnbind -s, 108
  with fncreate -s, 108
  with fnunbind, 108
name not found message, 103
name resolution
  context operation support requirements, 136 to 137
  DNS names, 62
  enterprise level namespace identifiers, 42
  status object information, 150
  X.500 names, 64 to 65
  XFN composite names, 163 to 166
  coexistence of explicit and implicit NNSPs, 165
  explicit NNSPs, 163 to 164
  implicit NNSPs, 164 to 165
  XFN links, 165

Index
name service (NS) resource records, 63
name services
  described, 3 to 4
  for printer context
    administration, 128 to 130
functions, 3
resource records, 63
namespace browser programming
  example, 168 to 177
code, 168 to 175
commands, 175
diagram, 168
sample output, 176 to 177
namespace identifiers
  canonical vs. Solaris customized, 42, 56
context creation, 74, 82
context ownership, 83
enterprise level, 41 to 42
  initial context bindings, 55
global level, 61
namespace policies, See policies
naming
  See also composite names; compound names; policies
context name inconsistencies, checking, 95 to 96
context operation names, 136
XFN attribute operations and, 144 to 145
XFN interface conventions, 134
naming system boundaries and component separators, 161 to 163
strong separation, 161 to 162
weak separation, 162 to 163
navigating, See browsing
next naming system pointers (NNSPs)
  X.500 name resolution, 64
  XFN composite name resolution
coexistence of explicit and implicit NNSPs, 165
explicit NNSPs, 163 to 164
implicit NNSPs, 164 to 165
NIS+
  described, 8
  federating with global naming systems, 111 to ??
  obtaining NIS+ root reference, 111 to 112
  under DNS, 112 to ??
  under X.500, 114 to ??
FNS administration, 69 to 109
access control checking, 98 to 99
browsing FNS structures using NIS+ commands, 97
error messages, 100 to 104
FNS attribute management, 92 to 94
FNS context creation, 73 to 83
FNS context management, 83 to 92
maintaining consistency between NIS+ and FNS, 94 to 96
mapping FNS contexts to NIS+ objects, 96
replicating FNS service, 72
resource requirements, 70
setting up FNS namespace, 71 to 72
setting up NIS+ service for FNS, 70 to 71
troubleshooting, 104 to 109
FNS implementation, 8
FNS policies, 30 to 32
  NIS+ domains and FNS organizational units, 30 to 31
  NIS+ hosts and FNS hosts, 32
  NIS+ users and FNS users, 31
printer context administration, 129 to 130
root reference, 111 to 112
NIS, printer context
  administration, 129
NIS_GROUP environment variable, 70 to 71
niscat command
  checking access control, 98 to 99
  listing fns_hosts contents, 97
nischgrp command, 99
nischmod command, 99
nischown command, 99
nisls command
  browsing FNS structures, 97
  verifying ctx_dir creation, 71
nismkdir command
  master server assignment, 71
  replicating FNS service, 72
nisping command, 72
NNSPs, See next naming system pointers (NNSPs)
no permission message, 103, 105 to 106
no such attribute message, 103
no supported address message, 103
not a context message, 104
NS records, 63
nsid context type, 82
  See also namespace identifiers

O
-0
  fnattr option, 94
  fnbind option, 89
-0, fncreate option
  all users context, 77
  organization context, 76
  overview, 74
  single user context, 78
objects
  classes, 195 to 199
  destroying named objects, 92, 107, 109
operation not supported
  message, 104
operations, See attribute operations;
  context operations; specific operations
org context type, 75
org namespace identifier
  initial context binding, 55
  “shorthand” binding, 59
organizational units
  composite name examples, 29
  contexts, 46 to 47
  creating, 75
  ownership, 76
  described, 28
  double slashes in organization names, 45, 99
enterprise namespace, 38
enterprise namespace policies, 43
fnlist doesn’t list
  suborganizations, 106
namespace identifier, 41
NIS+ domains and, 30 to 31
trailing dot in organization names, 31, 38
trailing slash in organization names, 100
orgunit or _orgunit namespace identifier
  FNS policy, 43
  initial context binding, 55
  resolution, 41
  “shorthand” binding, 59
orgunit// namespace identifier, 45
OSF DCE UUID XFN identifier
  format, 154

P
parent contexts
  See also contexts
  of enterprise root, 45
  of files, 52
  of hosts, 50
  of organizational units, 46
  of printers, 53
  of services, 51
  of sites, 47 to 48
  of users, 49
parsing
  compound names, 156 to 158
  syntax attributes, 156, 158
  XFN standard syntax model, 156 to 157
  XFN composite names, 187 to 189
partial result returned message, 104
passwd.org_dir table, 31
periods, See dots (.)
permissions
  attribute no permission message, 101
  changing, 99
  checking, 98 to 99
  no permission message, 103, 105 to 106
pipe character (|) in BNF notation, 186
plus sign (+) in BNF notation, 186
pointer types, 135
  See also next naming system pointers (NNSPs)
policies, 25 to 35
  enterprise namespace, 37 to 60
    arrangement of objects, 28
    file namespace, 40
    host namespace, 39
    illustrated, 27
    initial context bindings, 54 to 60
    namespace identifiers, 41 to 42
    namespace structure, 42 to 54
    organizational unit
      namespace, 38
    printer namespace, 40
    service namespace, 39 to 40
    site namespace, 38
    table of policies, 43
    user namespace, 39
  global namespace, 61 to 65
    federating DNS, 62 to 65
    federating X.500, 63 to 65
  initial context bindings for global naming, 62
  goals, 25
  information not specified, 26
  information specified, 26
  levels of services, 26, 27
  need for uniformity, 6 to 7
  overview, 26 to 28
  principles for, 7 to 8
  target client applications, 32 to 35
  predefined constants, 134
  primary status code, 150
  principles for FNS policies, 7 to 8
  Print Tool, 32
printer context type
  administration
    using files, 128
    using NIS, 129
    using NIS+, 129 to 130
  creation, 80
printer namespace identifier
  FNS policy, 43
  resolution, 41
printers
  context administration
    using files, 128
    using NIS, 129
    using NIS+, 129 to 130
  context creation, 80
  contexts, 53
  enterprise namespace, 40
  enterprise namespace policies, 43
  FNS policies, 32
  FNS-based naming, 9
  namespace identifier, 41
  programming example, 178 to 183
    client, 179 to 181
    server, 181 to 183
Index

printing
See also listing
  references used for binding, 89, 90
programming, See application
  programming; client
  programming interfaces

Q
quotation marks
  BNF notation, 186
  XFN composite name syntax, 159
  XFN standard syntax model, 157

R
-" fncheck option, 95, 96
  fncreate option, 74
  fncreate_fs(1M) option, 121
-` fnbind option, 89
RAM, memory-management policies for
  client interfaces, 135
references
  address properties, 12
  bad reference message, 101
  binding composite names to, 89 to 91, 108
  defined, 11
  DNS text record format for XFN
  references, 191 to 194
  getting for contexts, 142
  handle construction operation, 137
  NIS+ root reference, 111 to 112
  printing references used for
    binding, 89, 90
  status object information, 150
  X.500 attribute syntax for XFN
  references, 195 to 199
  XFN interface parameters, 153 to 154
  registering service names, 40
  relative distinguished names, 13
removing
  bindings, 140 to 141
  cannot remove context, 107
  composite names from
    namespace, 91 to 92, 108 to 109
  deleting attributes or values, 93
  destroying
    context handles, 143
    named objects, 92, 107, 109
    subcontexts, 142
  renaming bindings, 91, 141
  replacing attributes or values, 93
  replicating FNS service, 72
requirements for FNS on NIS+, 70
resolution, See name resolution
resolver library, 62
resources
  insufficient resources
    message, 102
  requirements, 70
root reference for NIS+, 111 to 112
RPC services, FNS policies, 33

S
-" fnattr option, 93
  fnbind option, 89, 108
  fncheck option, 95
  fncreate option, 74, 108
separator character (/)
  FNS component separator, 42
  naming system boundaries and, 161 to 163
    strong separation, 161 to 162
    weak separation, 162 to 163
  XFN composite name syntax, 159 to 161
  XFN standard syntax model, 157
servers
NFS file servers, 118 to 119
NIS+
  directory, 71
  requirements for FNS, 70
  print server programming
    example, 181 to 183
service context type, 79 to 80
service or _service namespace identifier
  FNS policy, 43
  resolution, 41
services
  See also printers
    as enterprise policy entities, 28
    context creation, 79 to 80
    context ownership, 80
    contexts, 51 to 52
    enterprise namespace, 39 to 40
    enterprise namespace policies, 43
    levels, 26, 27
    namespace identifier, 41
    registering names, 40
sets of attributes, 155
setting up FNS
  FNS namespace setup, 71 to 72
  NIS+ service setup, 70 to 71
  replicating FNS service, 72
  resource requirements, 70
“shorthand” bindings, 59 to 60
single host context
  creating, 76
  troubleshooting creation, 107
single quote in XFN composite name syntax, 159
  See also quotation marks
single user context
  creating, 78 to 79
  troubleshooting creation, 107
site context type, 81
site or _site namespace identifier
  FNS policy, 43
  initial context binding, 56
  resolution, 41
  “shorthand” binding, 60
sites
  composite name examples, 30
  context creation, 81
  context ownership, 81
  contexts, 47 to 49
  enterprise namespace, 38
  enterprise namespace policies, 43
  as enterprise policy entities, 28
  namespace identifier, 41
slash (/)
/... namespace identifier, 62
  double slashes in organization names, 45, 99
  FNS component separator, 42
  in service names, 39
  trailing, in organization names, 100
  XFN component separator, 157, 159 to 161
Solaris
  customized namespace identifiers, 42, 56
  FNS implementation
    applications, 10
    DNS, 9
    file naming, 9
    NIS+, 8
    printer naming, 9
    X.500, 9
status codes, 150 to 153
  link status, 150
status objects, 150
  base attribute interface, 145
  base context interface, 137
storage requirements for FNS on NIS+, 70
strings
  composing XFN composite name strings, 189 to 190
  decomposing XFN composite name strings, 187 to 189
  XFN composite name syntax, 159 to 161
  XFN identifier formats, 154
  XFN interface parameters, 155
  XFN standard syntax model, 157
subcontexts, See subordinate contexts
subordinate contexts
  See also contexts
   creating, 141 to 142
   destroying, 142
   of enterprise root, 45 to 46
   of files, 53
   of hosts, 50 to 51
   of organizational units, 47
   of printers, 54
   of services, 52
   of sites, 48
   of users, 49
suborganizations not listed by fnlist, 106
success message, 104
syntax not supported message, 104

T
  -t
    fncheck option, 95
    fncreate option, 74, 107
    _t suffix, 134
thisens or _thisens namespace identifier
  host-related binding, 58
  initial context binding, 55
thisorgunit or _thisorgunit namespace identifier
  host-related binding, 58
  initial context binding, 55
thisuser namespace identifier
  initial context binding, 55
  user-related binding, 56
too many attribute values message, 104
trailing dot (.) in organization names, 31, 38
troubleshooting
  See also error messages
   contexts
     cannot create host- or user-related contexts, 107
     cannot remove context, 107
     checking naming inconsistencies, 95 to 96
     debugging creation, 74
      fndestroy does not return “operation failed”, 109
     fnlist doesn’t list suborganizations, 106
     fnunbind does not return “operation failed”, 109
     initial context
     cannot obtain, 104
     nothing in, 105
     “name in use” messages, 108
     “no permission” messages, 105 to 106
     status codes, 151 to 153
TXT records in XFN references, 191 to 194

U
  -U
    fnattr option, 94
    fnbind option, 89
    -u, fncheck option, 95
unavailable message, 104
underscore (_)
  in namespace identifiers, 41, 56
  in XFN terminology, 41

Index
UNIX hierarchical naming system, 13
updating
  bindings, 140 to 141
FNS namespace, 95
user context type, 78 to 79
user or _user namespace identifier
  FNS policy, 43
  initial context binding, 55
  resolution, 41
  “shorthand” binding, 59
username context type, 39, 77
user-related bindings, 56 to 57
users
  as enterprise policy entities, 28
  bindings for enterprise naming, 56 to 57
  composite name examples, 29
  context creation
    all users, 74, 77
    single user, 78 to 79
  troubleshooting, 107
context ownership, 78
contexts, 49
enterprise namespace, 39
enterprise namespace policies, 43
namespace identifier, 41
NIS+ and FNS, 31
view of FNS, 17 to 18

X
-x, fnbind option, 89
X.500 global directory service
attribute syntax for XFN
  references, 195 to 199
described, 9, 63
directory information base (DIB), 63 to 64
federating, 63 to 65
federating NIS+ under, 114 to ?
FNS implementation, 9
hierarchical naming system, 13
name resolution, 64 to 65
X/Open Federated Naming
attribute model, 143 to 144
client programming interfaces, 133 to 158
  See also base attribute interface;
  base context interface
abstract data types, 134
base attribute interface, 143 to 149
base context interface, 135 to 143
  conventions, 134
memory management
  policies, 135
overview, 133 to 134
parameters, 153 to 155
  parsing compound names, 156 to 158
status codes, 150 to 153
status objects, 137, 145, 150
supporting interfaces, 133 to 134
usage, 134

component separator and naming
  system boundaries, 161 to 163
strong separation, 161 to 162
weak separation, 162 to 163
composite names, 159 to 166
  naming system boundaries and component
  separators, 161 to 163
resolution, 163 to 166
syntax, 159 to 161, 185 to 190
compound-name syntax model, 156 to 157
described, 5
DNS text record format for XFN references, 191 to 194
FNS conformity, xvii, 5
FNS vs., 5
identifier formats, 154
links
  composite name resolution, 165
described, 15
displaying binding, 83
error messages, 103
lookup operation, 139 to 140
status object information, 150
XFN header file, 134
XFN library, 134
object classes, 195 to 199
X.500 attribute syntax for XFN references, 195 to 199
X.500 entry supporting reference attributes, 114
XFN object class, 195 to 199
XFN, See X/Open Federated Naming
XFN-supplement object class, 195 to 199