

Programmer's Reference Guide

iPlanet™ Portal Server 3.0

806-5248-01

May 2000

Copyright 2000 Sun Microsystems, Inc.. All rights reserved.

Sun, Sun Microsystems, the Sun logo, Java, iPlanet, iPlanet Portal Server, and all Sun, Java, and iPlanet-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. in the U.S. and other countries. Products bearing SPARC trademarks are based upon an architecture developed by Sun Microsystems, Inc. ICA is a registered trademark of Citrix Systems, Inc., GO-Joe and RapidX are trademarks of GraphOn Corporation, and pcAnywhere, ColorScale, and SpeedSend are U.S. registered trademarks of Symantec Corporation. Information subject to change without notice. Federal Acquisitions: Commercial Software -- Government Users Subject to Standard License Terms and Conditions

The product described in this document is distributed under licenses restricting its use, copying, distribution, and decompilation. No part of the product or this document may be reproduced in any form by any means without prior written authorization of the Sun Microsystems, Inc. and its licensors, if any.

THIS DOCUMENTATION IS PROVIDED "AS IS" AND ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT SUCH DISCLAIMERS ARE HELD TO BE LEGALLY INVALID.

Contents

Preface	13
Who Should Use This Book	13
How This Book Is Organized	13
Documentation	14
Using UNIX Commands	14
What Typographic Conventions Mean	15
Shell Prompts in Command Examples	16
Chapter 1 Overview of the APIs	17
Extending iPlanet Portal Server	17
The iPlanet Portal Server APIs	18
Which APIs to Use?	20
Understanding iPlanet Portal Server APIs	20
Identify Needed iPlanet Portal Server APIs	21
Content Provider API	21
Profile and Policy API	22
Session API	22
Log API	23
Integrating an Application with iPlanet Portal Server Software	23
Chapter 2 Session API	25
Session API Overview	25
Implementing the Session API	25
Using the Session API	25
Session API Transport Protocol	28
Session API Message Format	28
Session API Classes and Interfaces	29

Sample Session Code	30
Instructions for using the HelloServlet	30
Import the iPlanet Portal Server Classes	32
Sample Code	33
 Chapter 3 Profile and Policy API	35
Profile and Policy API Overview	35
Profile and Policy API Functionality	35
Implementing the Profile and Policy API	37
Profile and Policy API Classes and Interfaces	37
Interactions, Assumptions, and Dependencies	38
Exception Handling	38
Using the Profile and Policy API	39
Getting Attribute Values	39
Setting Attribute Values	39
Checking Policy (Using Boolean Privileges)	40
Checking Policy (Using List Privileges)	40
Import the iPlanet Portal Server Classes	40
Sample Code	41
 Chapter 4 Log API	43
Log API Overview	43
Implementing the Log API	44
iPlanet Portal Server Classes	44
Log API Functionality	45
Creating Logs	45
Deleting Logs	46
Writing to a Log	46
Reading from a Log	47
Log List Retrieval	48
Querying Logs	48
Sample Code	49
 Chapter 5 Content Provider API	51
Content Provider Overview	51
Content Provider Functionality	51
Using the Sample Providers	52
Compiling Sample Provider Code	52
Implementing the Content Provider API	53
Provider Sample Code	53

Chapter 6 Pluggable Authentication API	57
Pluggable Authentication API Overview	57
Authentication Process Overview	58
Understanding the <code>.properties</code> File	59
Writing a Pluggable Authentication Module	62
Requirements	62
Recommendations	63
About using helpers	63
Integrating the Module	63
Sample Code	65
Sample Properties File	65
Sample Login Module Source	66
Sample XML File	67
 Chapter 7 Single Signon	 71
Single Signon Overview	71
Special Cases	71
Instructions for using Single Signon	72
Command Line Example	72
Include the iPlanet Portal Server Classes	73
 Chapter 8 Using the Command Line Interface	 77
Command Line Interface Overview	77
How it Works	77
ipsadmin Command	78
Usage	78
Using ipsadmin	79
Importing a New Component	79
Creating a New Domain	80
Creating a New Role	80
Creating a New User and Assigning a Role	81
Reading (Getting) a Profile	81
Changing a Profile	82
Deleting a Profile	83
Sample Code	83
 Chapter 9 Using the iPlanet Portal Server APIs	 87
Instructions for using the HelloServlet	87
HelloServlet Properties	88
HelloServlet XML	88
Prints HTML Output	90
Setting Privileges	90

Attributes and Privileges	90
Initializing the Servlet	91
Session API Examples	91
HTTP Request and Response	91
Session Event	92
Get a Session	93
Profile API Examples	93
Modify an Attribute	93
Get User Profile	94
Policy Checking	94
Log API Example	95
Method handles Logging	95
 Chapter 10 HTML Templates	97
Setting up Login Pages for Different Domains	97
How Authentication Templates Work	97
Templates for Customizing the Authentication Pages	97
How Desktop Templates Work	100
Templates for Customizing the iPlanet Portal Server Desktop	100
 Appendix A HTTP/XML Interface	103
HTTP/XML Interface Overview	103
Exchanging Information Between the Client and the Server	104
XML DTDs	104
PLL Request Set DTD	105
PLL Response Set DTD	105
PLL Notification Set DTD	106
Naming Response DTD	106
Naming Request DTD	107
Naming Request XML	107
Naming Response XML	108
Session-Related DTD and XML	109
SessionNotification DTD	109
Session Request DTD	110
Session Response DTD	111
Session Request XML	112
Session Response XML	112
Profile and Policy-related DTD and XML	113
Getting an Attribute Value Using XML	115
Log-related DTDs	115
DTD for Log API Communication	116

Appendix B Putting Code Together	117
Building an iPlanet Portal Server Provider	117
Define Specific Requirements and Functionality	118
Identify non-iPlanet Portal Server Functionality	118
Define Application Attributes/Privileges	119
Define the Provider to iPlanet Portal Server	119
Sample Code	121
 Appendix C iPlanet Portal Server API Exceptions	 133
Profile API Exceptions	133
Log API Exceptions	135
Session API Exceptions	136
 Glossary	 139
 Index	 149

List of Figures

Figure 1-1	End User Component Interaction Flow	18
Figure 2-1	Session Service Block Diagram	27
Figure 3-1	The Profile and Policy API Organization Structure	36

List of Tables

Table 1-1 The iPlanet Portal Server APIs.....	18
Table 6-1 Tasks to Customize Authentication	57
Table 6-2 The .properties File Directives	60
Table 10-1 HTML Template Files	98
Table B-1 Minimal Routines for a Provider	117
Table B-2 Sample Provider Attributes.....	119
Table B-3 Sample Attributes and Privileges	121
Table C-1 Profile API Exceptions	133
Table C-2 Logging API Exceptions	135
Table C-3 Session API Exceptions	136

The Programmer's Reference Guide explains how to use the iPlanet™ Portal Server 3.0 application programming interfaces (APIs), including the Java APIs and the over-the-wire (non-Java) APIs.

Who Should Use This Book

This document is intended for developers who are writing or modifying applications to communicate with the iPlanet Portal Server product.

How This Book Is Organized

This book contains the following chapters and appendices:

Overview of the APIs provides an introduction to some of the tools available

Session API defines applications to access session services provided by the Session Server.

Profile and Policy API provides a mechanism to manage user profiles and to impose policy.

Log API provides log management tools, and provides a set of Java classes so that the applications can create, retrieve, submit, or delete log information.

Content Provider API details the basics of developing new content modules for the desktop and introduces several sample providers.

Pluggable Authentication API describes requirements for writing a supplemental authentication module, and provides information about customizing the authentication pages.

Single Signon provides for session/user authentication at initial signon by the session server.

Using the Command Line Interface describes the command-line interface, and how to import XML files to register (or update) iPlanet Portal Server applications or content providers.

Using the iPlanet Portal Server APIs gives samples of the APIs and how to use them.

HTML Templates contains information to make substantive changes to layout or design of browser pages, or to add extra functionality.

HTTP/XML Interface details the processes and issues involved in communicating with the iPlanet Portal Server applications using the exposed HTTP/XML interface.

Putting Code Together describes the development process for a sample iPlanet Portal Server desktop provider application that touches on the public APIs available for integrating an application with the iPlanet Portal Server desktop.

iPlanet Portal Server API Exceptions lists the exceptions generated during an error in the Profile, Logging, and Session APIs.

Documentation

iPlanet Portal Server 3.0 documentation includes:

- *iPlanet Portal Server 3.0 Installation Guide*
- *iPlanet Portal Server 3.0 Administration Guide*
- *iPlanet Portal Server 3.0 Programmer's Reference Guide* (this book)
- *iPlanet Portal Server 3.0 Release Notes*
- Online help for users and online help for system administrators
- `http://yourserver:port/docs/en_US/javadocs`

Using UNIX Commands

This document contains some information on basic UNIX® commands and procedures. For more information outside of this document, see the following:

- AnswerBook2™ online documentation for the Solaris™ software environment.

- Browse for a specific book title or subject at:
`http://docs.sun.com`

What Typographic Conventions Mean

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

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

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.

Shell	Prompt
C shell prompt	machine_name%
C shell superuser prompt	machine_name#
Bourne shell and Korn shell prompt	\$
Bourne shell and Korn shell superuser prompt	#

Overview of the APIs

The Programmer's Reference Guide documents the public Java application programming interfaces (APIs) that are included in the iPlanet Portal Server product, as well as documents the exposed HTTP/XML interfaces.

The Programmer's Reference Guide offers information to any programmer customizing iPlanet Portal Server software. For example, use the Session and the Profile APIs to integrate the application with the iPlanet Portal Server software and utilize single signon capabilities, or the Content Provider modules for the user desktop.

NOTE Detailed information on APIs is available in the Javadocs from
`http://yourserver-name:port/docs/en_US/javadocs/`

Extending iPlanet Portal Server

iPlanet Portal Server can be extended in several ways, as illustrated in Figure 1-1 on page 18. If additional authentication capabilities are needed use the Pluggable Authentication API to create them. To add Java based applications, use the Session, Profile and Policy, and Log APIs to integrate them into the iPlanet Portal Server framework. Finally, to have additional content providers in the iPlanet Portal Server desktop, use the Content Provider API (and optionally other APIs) to integrate the providers directly into the iPlanet Portal Server desktop.

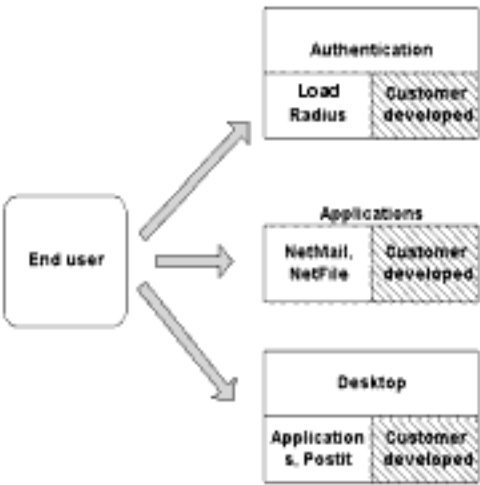


Figure 1-1 End User Component Interaction Flow

The end user interacts equally with the iPlanet Portal Server components as configured by default and with additional components written to integrate with the iPlanet Portal Server software.

The iPlanet Portal Server APIs

Table 1-1 summarizes the APIs in iPlanet Portal Server software.

Table 1-1 The iPlanet Portal Server APIs

API Name	Description
Profile and Policy	<p>The Profile allows applications to access profile information, with the Profile and Policy API, stored on the iPlanet Portal Server. Profile information includes user preferences, application attributes, platform-wide attributes, and configuration information. Applications may define and store their own application-specific attributes and configuration with this API.</p> <p>The Policy provides some methods, in the Profile and Policy API, to determine user privileges.</p>

Table 1-1 The iPlanet Portal Server APIs (*Continued*)

API Name	Description
Session	The Session API allows applications to verify whether a user has a valid session. When a user authenticates with the iPlanet Portal Server product, a session is created and stored on the iPlanet Portal Server. The Session API allows any application to validate the user's session and retrieve information about the session, such as the session state, the time remaining, and the user name.
Logging	The Log API allows applications to create, delete, write, and read logs and log records to and from the iPlanet Portal Server.
Content Provider	The Content Provider API provides methods for integrating content with the iPlanet Portal Server desktop.
Pluggable Authentication	The Authentication API allows the writing of a Java-based pluggable authentication module for iPlanet Portal Server software.

Writing programs that directly use the Java API is a far simpler and more efficient solution than using the exposed HTTP/XML interface, because the Java API “hides” the over-the-wire protocols, thus making implementation more straightforward.

The classes that can be included in the Java implementation automatically handle communication and data transfer with the iPlanet Portal Server. When considering a non-Java application, see Appendix A for a description of the processes, the communications protocols, the Document Type Definitions (DTDs), and other issues associated with applications that use the exposed HTTP/XML interface. Developers writing programs that will communicate with the exposed HTTP/XML interface to iPlanet Portal Server need to understand and be able to use eXtensible Markup Language (XML) and HTTP.

Which APIs to Use?

The iPlanet Portal Server APIs fall into several broad categories:

To do:	Use these APIs:
Application Development	Optionally use Profile and Policy, Session, and Logging.
Authentication Extension	Authentication, and optionally use Logging.
Desktop Extension	Provider, and optionally use Profile and Policy, Session, and Logging.

Understanding iPlanet Portal Server APIs

This section describes the development process for a sample iPlanet Portal Server desktop provider application that touches on the public APIs available for integrating an application with the iPlanet Portal Server desktop.

The following main steps outline the process:

1. Define high-level application requirements.
2. Determine which iPlanet Portal Server APIs support the high-level requirements.
3. Define the iPlanet Portal Server attributes.
4. Define the privileges that determine the policy for the application.
5. Create an XML file to define the provider to the iPlanet Portal Server desktop.
6. Import the XML file to the Profile server with `ipsadmin`.
7. Configure and modify the applications through the Administration Console.

A non-Java iPlanet Portal Server application cannot directly use the Java APIs, it must use the HTTP/XML interfaces that are defined in Appendix A, “HTTP/XML Interface”. Each XML DTD specifies the content and format of the information that can be sent to and received from the iPlanet Portal Server services.

Identify Needed iPlanet Portal Server APIs

Applications can interact with as many or as few of the APIs as necessary to provide functionality. These application may be configured by the administrator, either at a domain/role level or at the user-level, and by the user through the standard desktop interface. Additionally, the configuration may be retained between sessions. The application may also log the configuration settings of a user every time they are changed, recording accesses to a corporate database, or any other legitimate logging function, or if there is no logging requirement then there is no need to use the Log API.

Content Provider API

The application may use this provider to:

- Display information in the provider's portion of the desktop
- Display a page to the user that allows setting configurations
- Handle the submission of the configuration parameters

Additional methods are provided to:

- Query information about the provider
- Set information for the provider

To use the default actions, just create a provider that will be imported into (registered with) the iPlanet Portal Server product.

Registering a provider with the iPlanet Portal Server product requires creating an XML file with the specific attributes and privileges for this provider and the “common” attributes that a provider may have to utilize the default methods.

To import the XML into the Profile server use the `ipsadmin` command, and then from the Administration Console register the provider to the iPlanet Portal Server software.

Profile and Policy API

Because a user's configuration should be retained between sessions, it is advantageous to store configuration parameters in the Profile server. To do this, define appropriate attributes/privileges to be imported via the provider's XML file and ipsadmin. These attributes and privileges can be retrieved and modified through the Administration Console or the Profile and Policy API.

It is not a requirement to store anything in the Profile server, because a separate database or even just a flat file on the iPlanet Portal Server could be used, but using the profile server has some advantages:

- The profile server eliminates the need to create a new database and write access routines
- The profile server is hierarchal in nature, making it easy to set configuration parameters at a domain or role or user level
- In an iPlanet Portal Server configuration, with multiple servers, the Profile and Policy API will access the common profile server. When writing applications that use the Profile and Policy API, file locking and some database access functions are handled “behind the scenes” by the Profile and Policy API
- Simple “out of the box” administration of the provider's attributes and policy

Session API

The Session API is used to access information about a session. An iPlanet Portal Server session contains the information needed to get a profile from which to get/set attributes and will optionally check the policy to validate the iPlanet Portal Server user.

The Session API also provides methods to get:

- Client ID
- The domain
- Session type
- Idle time
- Session time

Log API

It is possible to write to a log file separate from the iPlanet Portal Server Log API, but using the Log API has some advantages:

- The Logging API provides some other useful log-management functions. For example, controlling the number and size of logs, that would have to be redeveloped if the API was not used
- The logs can be administered through the Administration Console, and access to the logs for any user is controlled by the Profile and Policy API

Integrating an Application with iPlanet Portal Server Software

When writing an application, privileges can be created to be associated with users and attributes and then be associated with uses and applications. For example, in a payroll application, create a privilege associated with viewing salary, or with changing salary, and create an attribute for a particular background color.

An overview of the sequence to integrate an application with iPlanet Portal Server software is as follows:

1. Identify data to be stored in the iPlanet Portal Server profile versus data that is to be obtained from other sources.
Payroll data is probably stored in a payroll database, and it would not be appropriate or necessary to migrate this to the iPlanet Portal Server profile database. However, there is probably authentication information that the user must enter to access the payroll database. For example, a user name and password, could be defined as attributes in the iPlanet Portal Server profile. See Chapter 7, “Single Signon”.
2. Decide which policy to assign for users whom access this application, and define corresponding privileges and default values.
3. Create an XML file that defines the data identified in Step 1 and Step 2.
4. Use the iPlanet Portal Server `ipsadmin` utility on the command line to import the XML and configure the Profile server to recognize the application.
5. Modify the application to use the iPlanet Portal Server APIs to identify the user rather than prompting for authentication information, when using a single-signon solution. This assumes that the application is already web-based. The code should:

- a.** Use the Session API to translate an HTTP request from the user into an iPlanet Portal Server session.
 - b.** Use the Profile and Policy API to access the application-specific authentication information that is stored in the iPlanet Portal Server profile.
- 6.** Use the Administration console to provide access to the payroll application.
- 7.** Use the Administration console to configure the policy for different roles.

Session API

Session API Overview

The Session Application Programming Interface (API) defines applications to access session services provided by the Session Server. Java applications can access session services by using the Java Session API.

Additionally, the Session API provides an XML DTD to define the format for data streams to provide to the server session process and to define the format for data streams coming from the server session process. These formats are required to access session functions from non-Java client software, but can be transparently integrated into Java applications, as shown in the sample code in this chapter.

Implementing the Session API

Using the Session API

A session represents a connection between a client and a server where information is exchanged between the two entities. It is critical to maintain state information between the two entities to prevent unauthorized clients from accessing resources in the iPlanet Portal Server platform. A state object, called a cookie, is used to maintain and store state information.

Sessions are a general mechanism which server side connections can use to both store and retrieve information on the client side of the connection. The addition of a simple, persistent, client-side state significantly extends the capabilities of Web-based client/server applications. A server, when returning an HTTP object to a client, may also send a piece of state information which the client will store.

Included in that state object is a description of the session credentials for which that state is valid. Any future HTTP requests made by the client which fall in that range will include a transmittal of the current value of the state object from the client back to the server.

There are two main types of sessions:

- User session
- Application session

A user session is associated with a user. An application session is associated with an application without the context of a user. The session type (user or application) property in a session is used to distinguish a user session from an application session.

A session is created when a user or an application authenticates itself successfully. The authentication service creates a new session in the iPlanet Portal Server platform through a private interface provided by the Session Service. An active session at minimum has the following properties:

Session ID	A random string generated by session service to uniquely identify the session. The string is carried in every HTTP/HTTPS request headers as cookies.
Session type	Whether this session is a user session or an application session.
Client id	The user id or the application id depending on the session type.
Domain name	The domain name of the user/application they belong to. It is used to distinguish users/applications of the same name. The domain name property is where the user's profile is located in the role tree.
Creation time	When the session is created.
Access time	The latest time the session is accessed.
Session state	Whether the session is valid or invalid.

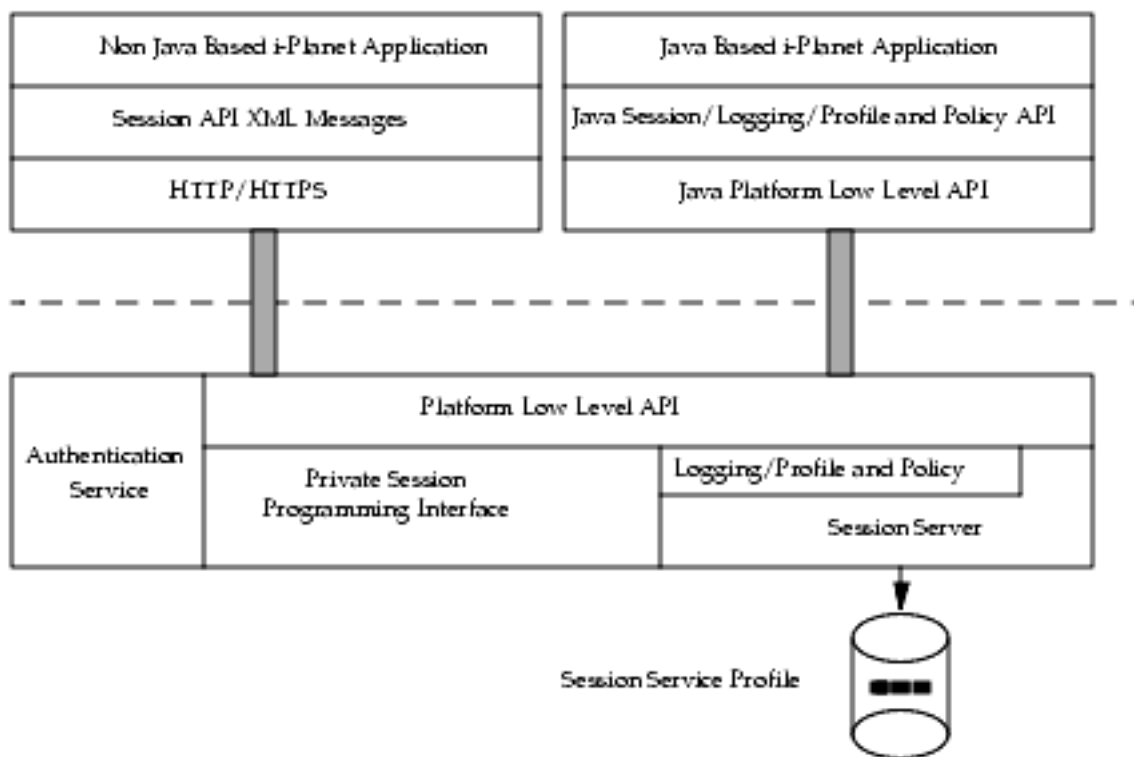


Figure 2-1 Session Service Block Diagram

The Session API can access session services provided by the iPlanet Portal Server software by using over-the-wire protocol. This protocol consists of the transport protocol and detailed message format in order to access iPlanet Portal Server services. A Java implementation of this over-the-wire protocol is also provided so that the transport protocol and message format details can be hidden from Java application developers.

Non-Java applications access session services by using HTTP/HTTPS transport protocol and XML messages defined by the Session API to communicate with the session server.

Session API Transport Protocol

As HTTP is the main communication protocol in the iPlanet Portal Server platform and well defined, there is no need to invent a new syntax and semantics for the transport of the Session API, it is a natural choice to use HTTP as the transport protocol to access those session services for the Session API.

Session API Message Format

All session requests, responses, and events are encoded to XML. The main advantage to use XML encoded message is that non-java applications can access session services of the iPlanet Portal Server platform by using the required transport protocol and XML message format described in this section.

The following are the main session requests used by the Session API:

- Get a session
- Get all valid sessions. (Protected by policy)
- Destroy a session. (Protected by policy)
- Logout a session
- Add a listener on a session
- Set session properties

A session can be destroyed by an administrator. It will also be destroyed based on the session idle time and session maximum duration time.

Idle time	The difference between the current time and the last access time.
Session duration time	The difference between the current time and the session creation time.

The default maximum session idle time and duration time shall be in the Session Service Profile, which makes it possible to assign different maximum idle and duration time to different users and applications by overwriting those values in the users profiles and applications profiles respectively.

Session API Classes and Interfaces

The classes that can be included in the Java implementation automatically handle communication and data transfer with the iPlanet Portal Server product.

Session Class	This class represents a session. It contains session related information such as session id, session type (user/application), client id (user id or application id), session creation time, latest session access time, and session state. It also allows applications to add listener for session events.
SessionID Class	This class is used by applications to identify individual iPlanet Portal Server sessions. The SessionID information includes the originating domain of the user.
SessionEvent Class	This class represents a session event. It contains the session object for the event, event type, event time, and event specific information corresponding to the event type if any.
SessionListener Interface	This is an interface which needs to be implemented by applications in order to receive session events.

Sample Session Code

The following code sample illustrates how a new application might use the Session API.

Instructions for using the HelloServlet

1. Set `IPS_BASE` to the iPlanet Portal Server installation directory.
2. Change directory and make the file as shown in the following example:

```
# cd $IPS_BASE/SUNWips/sample/api
# make
```

3. Copy the class files to the appropriate directory on the portal server under:

```
$IPS_BASE/SUNWips/lib
```

For example, all class files would be copied to:

```
$IPS_BASE/SUNWips/lib/com/iplanet/portalserver/api
```

4. Modify the web server configuration.

The web server configuration files are in the directory:

```
$IPS_BASE/netscape/server4/https-servername/config
```

where *servername* is the FQDN of the portal server.

5. Add the following line to the web server `servlets.properties` file:

```
servlet.helloservlet.code=com.ipplanet.portalserver.api.HelloServlet
```

Replace the package and servlet names with the names that were chosen for this HelloServlet.

6. Add the following line to the web server `rules.properties` file:

```
/helloservlet=helloservlet
```

7. As root, Import `iwtHelloServlet.xml` using `ipsadmin`, as shown in the following example:

```
# $IPS_BASE/SUNWips/bin/ipsadmin -import iwtHelloServlet.xml
```

8. copy file `iwtHelloServlet.properties` to `$IPS_BASE/SUNWips/locale` directory
9. Restart the iPlanet Portal Server server.

```
# /etc/init.d/ipsserver start
```

10. Test the servlet by logging in to the iPlanet Portal Server desktop and entering the following URL:

```
https://gateway/http://server:8080/helloservlet
```

where *gateway* and *server* are replaced by the names of the gateway and server.

Import the iPlanet Portal Server Classes

At a minimum, the Java client application should import the iPlanet Portal Server Profile, logging, and Session classes, as shown here.

Code Example 2-1 Importing iPlanet Portal Server Classes

```
// @(#)HelloServlet.java 1.1      00/04/20 Copyright (c) 1999 Sun
Microsystems, Inc., All rights reserved.

package com.iplanet.portalserver.api;

import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.Vector;

import com.iplanet.portalserver.session.*;
import com.iplanet.portalserver.profile.*;
import com.iplanet.portalserver.logging.*;
import com.iplanet.portalserver.util.*;
```

While directly access the classes as needed, importing the logging and session classes will allow better use of the Session functions.

The sections below briefly describe some of the functionality available, but reference the Javadocs online at:

http://yourserver:port/docs/en_US/javadocs

Sample Code

The following code sample illustrates how a new application might use the Session API.

Code Example 2-2 Sample Session API

```
public class HelloServlet extends HttpServlet {

    public void doGet(HttpServletRequest req, HttpServletResponse
res)
throws ServletException, IOException {
        try {
            // Get a session
            SessionID sid = new SessionID(req);
            Session sess = Session.getSession(sid);

            // Validate a session
            if (sess.getState(false) != Session.VALID)
                return;

            // Add a session listener
            sess.addSessionListener(new HelloSessionListener());

        } catch (SessionException e) {
        }
    }

    class HelloSessionListener implements SessionListener {

        public void sessionChanged(SessionEvent e) {
            Session sessionEvt = null;

            // if the session is still valid, just return
            // without doing anything
            try {
                sessionEvt = e.getSession();
                if (sessionEvt.getState(false) == Session.VALID)
                    return;
            } else {
                // clean up profile before quitting
            }
        } catch (Exception se) {}
    }
}
```


Profile and Policy API

Profile and Policy API Overview

The Profile and Policy Application Programming Interface (API) provides developers of iPlanet Portal Server client software a mechanism to manage user and role profiles. It also allows application programmers to perform policy checks on the user before granting any permission. Additionally, the iPlanet Portal Server Administration console relies on the Profile and Policy API to manage users and roles.

Additionally, the Profile and Policy API provides an XML DTD to define the format for data streams to provide to the server profile process and to define the format for data streams coming from the server profile process. These formats are required to access profile and policy functions from non-Java client software, but can be transparently integrated into Java applications, as shown in the sample code in this chapter.

Profile and Policy API Functionality

The Profile and Policy API performs a variety of profiling and access control tasks within the iPlanet Portal Server environment, including:

- Returning any or all attributes and values to the calling application
- Uses the policy methods to check user access privilege before granting any permission

All of the main entities controlled by the Profile and Policy API (users, roles, and domains) are organized in a tree structure, as shown in Figure 3-1. Each entity inherits attributes from its parent and separately maintains its own attributes, which override inherited attributes.

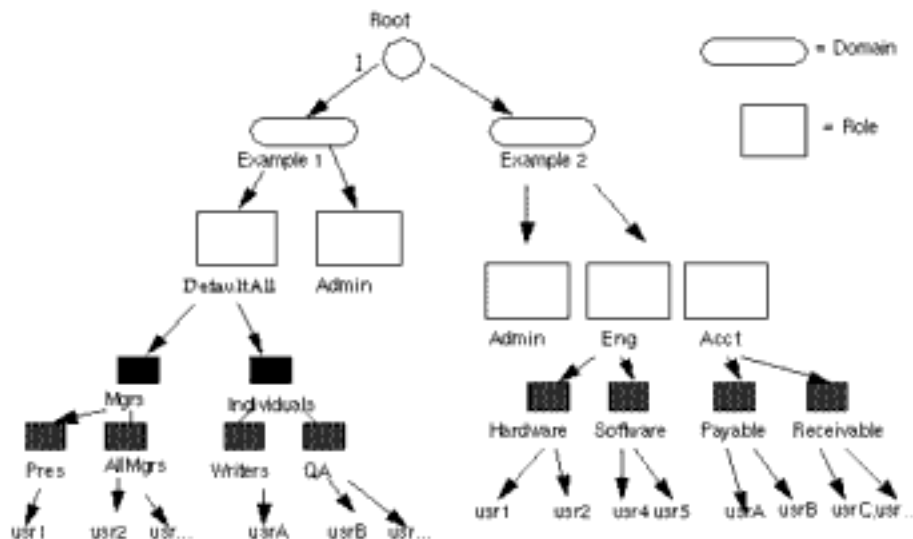


Figure 3-1 The Profile and Policy API Organization Structure

Profile and Policy attributes are stored in the form of name-value pairs in profile database. Access privileges are special attributes stored in the Profile and Policy database. Boolean type privileges have a boolean value of true or false. List type privileges have an allow value list and a deny value list.

The policy for a domain, role or user is implemented by setting the privileges for an individual domain, role, or user profile.

Implementing the Profile and Policy API

As with the other APIs, implementing an iPlanet Portal Server client application in Java is substantially less complex than implementing in any other language, simply because referencing existing iPlanet Portal Server classes masks much of the communication and protocol manipulation. Therefore, only be concerned with the Profile attributes and values wanting to create, write to, read from, or otherwise manipulate, and can ignore the communication protocols involved for all Java client implementations.

Profile and Policy API Classes and Interfaces

The Profile and Policy API provides the following Java classes and interfaces:

- **Profile Class** provides the following methods:
 - **Profile methods** provides common methods to create, delete, and access profile attributes
 - **Policy methods** provides methods to check access privileges
- **ProfileEvent Class** represents profile event notification. This notification is generated when profile attribute, or privilege is changed
- **Profile Exception Class** is a generic profile service exception For simplicity this profile API throws this exception with a specified exception type. See Appendix C, “iPlanet Portal Server API Exceptions”
- **Profile Listener (Interface)** This interface needs to be implemented by the applications in order to receive profile events

Specific implementation details are contained in the Profile and Policy API Javadocs, available online at:

`http://yourserver:port/docs/en_US/javadocs`

The following section outlines the procedures for using the Profile and Policy API methods and classes.

Interactions, Assumptions, and Dependencies

The Profile and Policy API uses:

- The Session API to validate the user session
- Platform low level API for over the wire communication

See Appendix A for more information about direct communication with the profile server process.

Additionally, it is assumed that applications supply lists of profile attributes, access privileges, and their initial values.

Exception Handling

The Profile and Policy API performs a wide range of checks and throws exceptions in the following cases. See Appendix C, “iPlanet Portal Server API Exceptions” for all the Profile API exceptions.

- Requested profile is not found
- Failure of store operation
- User does not have permission to do requested operation on an attribute
- Requested attribute is not found in user profile
- User session is not valid/inactive
- Privilege not found in user profile
- Invalid value supplied for attribute
- Illegal privilege name
- Illegal attribute name
- Illegal wild character expression
- Illegal wildcard expression
- Illegal match value

Using the Profile and Policy API

The Profile and Policy API provides methods to add and delete roles to the permission lists.

Each attribute defined in the Profile database has its own qualifiers, including:

- Read/Write permission Lists
- Remote flag

Read/Write permission Lists tell which role can perform read/write operations on an attribute.

Getting Profile Object

For example, an application programmer could get a profile object with the Session API, as follows:

```
Profile p = session.getUserprofile ( ) ;
```

Getting Attribute Values

An application programmer could get a profile attribute with the Profile and Policy API, as follows:

```
string name = p.getAttributeString ( "HelloServlet-color" ) ;
```

- Returns `HelloServlet-color` attribute **value**

Setting Attribute Values

For example, an application programmer could set a profile attribute with the Profile and Policy API, as follows:

```
p.getAttributeString ( "HelloServlet-color", "blue", Profile.NEW ) ;
```

- Sets `HelloServlet-color` to **blue**

Checking Policy (Using Boolean Privileges)

For example, an application programmer could check policy with the Profile and Policy API, as follows:

```
p.isAllowed("HelloServlet-execute")
```

- Returns **true** if `HelloServlet-execute` is set to **true**
- Returns **false** if `HelloServlet-execute` is set to **false**

Checking Policy (Using List Privileges)

For example, an application programmer could set and check policy with the Profile and Policy API, as follows:

```
p.isAllowed("HelloServlet-changeColor", session.getClientDomain(), Profile.Regular)
```

- Returns **true** if user domain is in **allow list** and **not in deny list**
- Returns **true** if allow list contains '*' and user domain is **not in deny list**
- Returns **false** if **user domain** is in **deny list**
- Returns **false** if **user domain** is **not in deny list** or **allow list**

Refer to Code Example 3-2 for a sample Profile API.

Import the iPlanet Portal Server Classes

At a minimum, the Java client application should import the iPlanet Portal Server Profile, and Session classes, as shown here.

Code Example 3-1 Importing iPlanet Portal Server Classes

```
import java.io.*;
import java.util.*;
import java.net.*;
import com.iplanet.portalserver.naming.*;
import com.iplanet.portalserver.session.*;
import com.iplanet.portalserver.profile.*;
```

Sample Code

The following code sample illustrates how a new application might use the Profile API. This code segment uses the *Session* object to get the user profile, which will then check a privilege. If the provider does not throw an exception, an attribute is returned.

Code Example 3-2 Sample Profile API

```
public class HelloServlet extends HttpServlet {

    public void doGet(HttpServletRequest req, HttpServletResponse
res)
    throws ServletException, IOException {
        try {

            // Get a session as described in the previous sample :
            sess

            // Get user profile name from the session object
            Profile p = s.getUserProfile();

            // Get a profile attributes

            String color =
p.getAttributeString("HelloServlet-color");
            String name = p.getAttributeString("HelloServlet-name");

            // Add a profile listener
            p.addProfileListener(new HelloProfileListener());

            // Get policy information
            if (p.isAllowed("HelloServlet-execute")) {
                System.out.println("User is allowed to execute this
program");
            }
        }
    }
}
```

Code Example 3-2 Sample Profile API (*Continued*)

```

        // The HelloServlet-changeColor is defined as a list type
        privilege.
        // What's in the list is domains. If the user domain is
        in the
        // privilege HelloServlet-changeColor's allow list, the
        user is allowed
        // to change color. Otherwise, the user is denied to
        change color.

        if (p.isAllowed("HelloServlet-changeColor",
s.getClientDomain(), Profile.REGULAR)) {
            System.out.println("User is allowed to change color");
        }
    } catch (ProfileException e) {
    }
}

public class HelloProfileListener implements ProfileListener {
    public void profileChanged(ProfileEvent notify){

        Profile p = notify.getProfile();
        int type = notify.getType();

        // Either the color or the name attribute may have changed
        // Get the new values for these attributes.
        if (type == ProfileEvent.PROFILE_CHANGE) {

            try {
                String color =
p.getAttributeString("HelloServlet-color");
                String name =
p.getAttributeString("HelloServlet-name");
            } catch (ProfileException pe) {
                System.out.println("Profile: getAttribute() failed");
            }
            return;
        } else {
            // no attributes were changed
            // profiles were created or deleted
            return;
        }
    }
}
}

```

Log API

Log API Overview

This chapter provides an overview of Log Application Programming Interface (API) which can be used by the iPlanet Portal Server applications to perform logging activities.

The Log API provides log management tools for the iPlanet Portal Server applications, and provides a set of Java classes so that the applications can create, retrieve, submit, or delete log information.

Some of the information that can be tracked and recorded by the Log API include:

- User identification and actions taken
- System usage, access time, and failures

Additionally, the Log API provides an XML DTD to define the format for data streams to provide to the server logging process and to define the format for data streams coming from the server logging process. These formats are required to access logging functions from non-Java client software, but can be transparently integrated into Java applications, as shown in the sample code in this chapter.

The logging server process resides on the same network as the iPlanet Portal Server application server and gateway. The log file format is ASCII text, so administrators may use any utilities available to them for viewing ASCII text files.

NOTE	The details of the Log API's methods are in the Log API Javadocs, available from the server at <code>http://yourserver:port/docs/en_US/javadocs</code>
-------------	--

Implementing the Log API

As with the other APIs, implementing a client application in Java is substantially less complex than implementing in any other language. By referencing existing iPlanet Portal Server classes to create, write to, read from, or otherwise manipulate logs, masks much of the communication and protocol manipulation.

The Log API provides:

Log Manager	A set of Java classes to an application so that the application can create or delete a log, submit or retrieve log information, query a log, and retrieve a list of available logs.
Log Record	A Java class to allow an application to fill in a single log request information from the application so that the application can submit such single log request information to a log file.

iPlanet Portal Server Classes

At a minimum, the Java client application should import the logging and session classes, as shown here.

Code Example 4-1 Importing iPlanet Portal Server Classes

```
import java.io.*;
import java.util.*;
import java.net.*;
import com.iplanet.portalserver.session.*;
```

While directly access the classes as needed, importing the logging and session classes will allow better use of the Logging functions.

The sections below briefly describe some of the functionality available, but reference the Javadocs online at:

http://yourserver:port/docs/en_US/javadocs

Log API Functionality

The Log API offers the following capabilities to applications:

- Creating Logs
- Deleting Logs
- Writing to a Log
- Reading from a Log
- Log List Retrieval
- Querying Logs

Creating Logs

Applications can create a log file by invoking the `LogManager` constructor and passing the requested log file name value to the `create` method.

Code Example 4-2 Create a New Log (Minimal Code)

```
LogManager logMgr = new LogManager(session);
try {
    logMgr.create("logname");
} catch (LogException e) {
    System.out.println("Log Creation fails: " + e);
}
```

`Create` throws an exception if:

- The filename is invalid
- The file already exists
- The application does not have permission to create files

Deleting Logs

Applications can delete a log by constructing an object of type `LogManager` and passing the log file name to the object's `delete` method.

Code Example 4-3 Delete a Log (Minimal Code)

```
LogManager logMgr = new LogManager(session);
try {
    logMgr.delete("logname");
} catch (LogException e) {
    System.out.println("Log Deletion fails: " + e);
}
```

`Delete` throws an exception if:

- The log does not exist
- The filename is invalid
- The application does not have permission to delete files

Writing to a Log

Applications submit *key-value pairs* to log records, to a log. First, the application must *create a log record*, then write to the log, and catch exceptions as required.

Code Example 4-4 Writing Records to a Log (Minimal Code)

```
LogManager logMgr = new LogManager(session);
LogRecord logrec = new LogRecord(key, value);
try {
    logMgr.write("logname", logrec);
} catch (LogException e) {
    System.out.println("Log info writing fails: " + e);
}
```

`LogException` is thrown if:

- Failure of a log record submission
- Privilege denied

- Target log does not exist
- Target log is inactive
- Invalid session
- Fatal system error

Reading from a Log

Applications retrieve information from a log and may selectively retrieve information with a query. See “Querying Logs,” on page 48.

Code Example 4-5 Reading Records from a Log (Minimal Code)

```
LogManager logMgr = new LogManager(session);

Vector loginfo = new Vector();
try {
    loginfo = logMgr.read("logname");
} catch (LogException e) {
    System.out.println("Log Info Retrieval fails: " + e);
}
```

A `LogException` is thrown if an error occurs as follows:

- Invalid session is encountered
- The reading privilege is denied
- The target log does not exist
- Other fatal system errors are encountered

Log List Retrieval

Applications can get a list of all log names in the system.

Code Example 4-6 Retrieve Existing Log List (Minimal Code)

```
LogMgr logMgr = new LogManager(session);
Vector llist = new Vector();
try {
    llist = logMgr.list();
} catch (LogException e) {
    //System.out.println("Log List Retrieval fails: " + e);
}
```

A `LogException` is thrown if an error occurs for the following reasons:

- Invalid session is encountered
- The listing privilege is denied
- The log list does not exist
- Other fatal system errors are encountered

Querying Logs

To retrieve log records from a log with a query, the log should include such information as:

- A valid and existing log name.
- A query string.

Code Example 4-7 Query Log Information (Minimal Code)

```
LogManager logMgr = new LogManager(session);

vector list = new vector()
try {
    vector loginfo = logMgr.read("logname", "domain = iplanet.com");
}
catch (LogException e) {
    System.out.println("Log Info Retrieval fails: " + e);
}
```


This queries the specified log with the boolean query provided. For example, query for “domain = sun.com”.

A `LogException` is thrown if it an error occurs as follows:

- Invalid session is encountered
- The reading privilege is denied
- The target log does not exist
- Or other fatal system errors are encountered

Sample Code

The following sample code illustrates a basic implementation of logging functionality.

Code Example 4-8 Sample Log API

```

LogManager lm = new LogManager(session);

//create a log
try {
    lm.create("xyz");
} catch (LogException e) {
    System.out.println("e");
}

//submit log information
try {
    for (int i=0; i<10; i++) {
        LogRecord lr = new LogRecord("typehello", "msghello");
        lm.write(lr, "xyz");
    }
} catch (LogException e) {
    System.out.println("e");
}

//retrieve log information
try {
    Vector r = new Vector();
    r = lm.read("xyz");
    for (int i=0; i<r.size(); i++) {
        System.out.println(r.elementAt(i));
    }
} catch (LogException e) {
    System.out.println("e");
}

//delete a log

```

Code Example 4-8 Sample Log API (*Continued*)

```
try {
    lm.delete("xyz");
}
catch (LogException e) {
    System.out.println("e");
}

//query log information
try {
    loginfo = logMgr.read("xyz", "domain = iplanet.com");
}
catch (LogException e) {
    System.out.println("e");
}

//obtain a list of log names in the system
try {
    Vector ll = new Vector();
    ll = lm.list();
    for (int i = 0; i < ll.size(); i++) {
        System.out.println( ll.elementAt(i) );
    }
}
catch (LogException e) {
    System.out.println("e");
}
```

Content Provider API

Content Provider Overview

The iPlanet Portal Server software allows multiple sources of information, applications, and services to be displayed within a single page or set of pages that the user can view in a browser. The page in which the content is included is known as the desktop. The various sources of content are displayed in rectangular areas arranged in rows and columns within the desktop called channels. A Java class, called a provider, is responsible for converting the content in a file, or the output of an application or service into the proper format for a channel. A number of providers are shipped with the Portal Server including a bookmark provider, an application provider, and a notes provider. As the desktop is imaged, each provider is queried in turn for the content of its associated channel. Some providers are capable of generating multiple channels based upon their configuration. This chapter details the basics of developing new content channels for the desktop and introduces a sample provider.

Content Provider Functionality

The iPlanet Portal Server desktop is designed to have four possible content layouts, with various combinations of 'thick' and 'thin' columns. The main desktop layout template consists of basic HTML to define the look (colors, logo, etc.) of the page, and tags designed to be replaced with the content for each of the columns.

The content for the columns will be provided by providers, which are custom Java classes. The Java classes are invoked by the desktop Servlet and the output is inserted into the output of the entire desktop for display. Each provider may define the look of the content section of part, or, the whole layout.

Channels can define their width as either thick or thin. This loosely defines the amount of horizontal screen real estate for the channel; a thick channel subjectively gets more horizontal screen real estate than a thin one. The absolute width is dependent upon the size of the browser window. A channel will expand veritically as needed; there is no restriction on vertical sizing of a channel. Providers designed for the 'thin' column should not be placed in the 'thick' column and vice-versa.

Using the Sample Providers

The iPlanet Portal Server package includes several sample Content Providers in the SUNWips package. The following sections explain the process to compile the providers, and add them to an existing iPlanet Portal Server installation.

See the *iPlanet Portal Server Administration Guide* for information on adding custom providers.

TIP See Appendix B, "Putting Code Together", an iPlanet Portal Server desktop provider sample application that touches on the public APIs available for integrating an application with the iPlanet Portal Server desktop.

Compiling Sample Provider Code

Before compiling the sample provider code, set certain environment variables:

- `setenv PATH $PATH:/usr/java/bin:/usr/ccs/bin`
- `JAVA_HOME` to the Java directory (usually `/usr/java`)
- `IPS_BASE` to the product installation directory (usually `/opt`)

To compile the Sample Providers, use the following process:

1. Change to the top level of the `samples` directory.

```
# cd /opt/SUNWips/samples
```

2. Make the samples:

```
# make
```

TIP Create a short shell script to more easily recompile the samples. The following sample works well with a default installation.

```
#!/bin/sh
JAVA_HOME=/usr/java
IPS_BASE=/opt
export JAVA_HOME IPS_BASE IPS_ROOT
cd $IPS_ROOT
make
```

Implementing the Content Provider API

To use the Content Provider API, develop classes that implement or extend the `ProfileProviderAdapter`, or the `ProviderAdapter`.

NOTE Complete documentation for all methods and classes is in the Javadocs, available in the installation at:
http://yourserver:port/docs/en_US/javadocs

Provider Sample Code

The following sample code implements a minimal Provider (HelloWorld). Compile this code as above, then use the process documented above to add this provider to the Portal Server system.

Code Example 5-1 HelloWorld Content Provider

```

package com.iplanet.portalserver.providers.helloworld;

import java.lang.*;
import java.util.Map;

import com.iplanet.portalserver.providers.*;

/**
 * This class implements a provider that prints the hello world
 * message.
 * It does not use the iPS Profile Service.
 * <p>
 * This class
 * only overrides the methods in the class
 * <code>ProviderAdapter</code>
 * that it needs to customize. In a production environment, you
 * should
 * always implement all of the method in the
 * <code>Provider</code> interface.
 * <p>
 * When installing this class, you MUST register it with the
 * desktop under
 * the same name returned by the <code>getName()</code> method!
 */

public class HelloWorldProvider extends ProviderAdapter
implements Provider {

    public HelloWorldProvider() {

    }

    public StringBuffer getContent(Map m) throws
    ProviderException {
    StringBuffer content = new StringBuffer("Hello World!");

    return content;
    }

    public String getTitle() {
    return "Hello World Provider";
    }

    public String getDescription() {
    return "This provider says 'Hello World!'";
    }

    public String getName() {
    return "iwtHelloWorldProvider";
    }

    public String getBackgroundColor() {

```

Code Example 5-1 HelloWorld Content Provider (*Continued*)

```
        return "#CCCCC";  
    }  
}
```


Pluggable Authentication API

Pluggable Authentication API Overview

This chapter describes requirements for writing a supplemental authentication module to plug into iPlanet Portal Server and provides information about customizing the authentication pages.

Table 6-1 Tasks to Customize Authentication

If you want to	Do
Change login prompts	Edit the <code>.properties</code> file associated with the login screen that will be changed. See “Understanding the <code>.properties</code> File,” on page 59.
Add authentication capability	<ul style="list-style-type: none"> • Create <code>.properties</code> file for the authentication module. See “Understanding the <code>.properties</code> File,” on page 59. • Write and integrate an auth module. “Writing a Pluggable Authentication Module,” on page 62.
Selectively enable or disable auth modules	Refer to <i>iPlanet Portal Server 3.0 Administration Guide</i>
Customize prompts and appearance for specific Domains	Refer to <i>iPlanet Portal Server 3.0 Administration Guide</i> .

The authentication methods available in iPlanet Portal Server include the following:

- RADIUS
 - RADUIS

- RADIUS-to-ACE/Server
- RADIUS-to-SafeWord
- RSA Security, Inc., Security Dynamics
 - SecurID, ACE/Server, ACE/Agent, Security Dynamics
- Secure Computing, Inc.
 - SafeWord server and tokens
- S/Key
- UNIX/NIS
- NT
- LDAP
- Personal Digital Certificate
- Membership

TIP	Before beginning to write authentication modules, contact the e-Commerce Solutions sales representative to find out if the module needed has already been written and is available from internal resources.
------------	---

For additional authentication capabilities, use the information in this chapter to write a custom authentication module and integrate it into the Portal Server.

Authentication Process Overview

Custom iPlanet Portal Server authentication modules require two main components:

- The `.class` file to authenticate the user
- The `.properties` file to specify prompts the user will see and the information that will be passed to the authentication module

The `.class` file implements the authentication type that is added. Write the authentication module to override certain methods provided by the API.

For information about the methods the module must override, see “Requirements,” on page 62.

Understanding the .properties File

The .properties file is the configuration file for an authentication module. The file specifies the text, tokens, and password prompts for the login pages associated with the authentication module.

Name the authentication module's .properties file name with the base class name (no package name) and the extension .properties.

For example: Sample.properties.

This file must reside in /etc/opt/SUNWips/auth/default on the iPlanet Portal Server software.

Code Example 6-1 The Form for the Properties File

```
SCREEN
TIMEOUT 60
TEXT Sample Login Page
TOKEN Enter User Name:
PASSWORD Enter User Password:

SCREEN
TIMEOUT 30
TEXT Sample Login Page 2
TOKEN Enter Favorite Color
TOKEN Enter Secret Pin Number
PASSWORD Enter Challenge form
```

Table 6-2 discusses the directives that can be included in a .properties file.

Table 6-2 The .properties File Directives

Directive	Description
SCREEN	Each SCREEN entry corresponds to one authentication state (authentication HTML page). The authentication module can set which screen is next, or it can allow the iPlanet Portal Server's <code>auth servlet</code> progress through the screens sequentially.
TIMEOUT n	The TIMEOUT directive is used to ensure that users respond in a timely manner. If the time between when the page is sent and the user submits his response is greater than “n” seconds, a time-out page is sent.
TEXT	The TEXT directive is similar to a title for the page, and the text <code><xxx></code> appears at the top of the screen area provided for the auth module. Only one TEXT directive per SCREEN should be specified. If more than one is provided, then the last one is displayed.
TOKEN yyy	<p>The TOKEN directive equates to the following HTML:</p> <pre><P>yyy
<INPUT TYPE="TEXT" NAME=TOKEN0></pre> <p>The input field's name starts at TOKEN0 and increments with each TOKEN or PASSWORD directive specified per SCREEN.</p>
PASSWORD zzz	<p>The PASSWORD directive equates to the following HTML:</p> <pre><P>yyy
<INPUT TYPE="PASSWORD" NAME=TOKEN0></pre> <p>The input field's name starts at TOKEN0 and increments with each PASSWORD or TOKEN directive specified per SCREEN.</p>
IMAGE image-path	The IMAGE directive allows auth module writers to display a background image on each page.
<REPLACE>	<p>The REPLACE tag allows a module to substitute dynamic text for the text accompanying token and password descriptions.</p> <p>Used in conjunction with the <code>setReplaceText()</code> method.</p>

The specific directives included will depend on the requirements of the authentication method and the extent of customizing the appearance of the prompts and displays through the authentication process.

Each `SCREEN` entry corresponds to one authentication state or authentication HTML page. When an authentication session is invoked, one HTML page is sent for each state. In Table 6-2 on page 60, the first state sends an HTML page asking the users to enter a token and a password. When the users submit the token and the password, the `validate()` method is called. The module gets the tokens, validates them, and returns them. The second page is then sent and the `validate()` method is again called.

If the module throws a `LoginException`, an authentication failed page is sent to the user. If no exception is thrown, which implies successful completion, the users are redirected to their default page. The `TIMEOUT` directive is used to ensure that the users respond in a timely manner. If the time between sending the page and the response is greater than the `TIMEOUT` value, a time-out page is sent.

When multiple pages are sent to the user, the tokens from a previous page may be retrieved by using the `getTokenForState` methods. Each page is referred to as a state. The underlying authentication module keeps the tokens from the previous states until the authentication is completed.

Each authentication session creates a new instance of the authentication Java class. The reference to the class is released once the authentication session has either succeeded or failed.

NOTE	Any static data or reference to any static data in the authentication module must be thread safe.
-------------	---

Writing a Pluggable Authentication Module

The following sections discuss the requirements and recommendations to follow when writing a new authentication module.

Requirements

A pluggable authentication module for iPlanet Portal Server desktop must override certain methods and should adhere to specific naming conventions and standards for easy integration.

NOTE	For a list and description of the methods used to write the authentication module, see the JavaDocs at http://yourserver:port/docs/en_US/javadocs
-------------	--

- Extend `com.iplanet.portalserver.auth.server.Login`
- Override the `validate()`, `init()`, and `getUserTokenId()` methods

The `validate` method replaces the input gathering method. Each time the user submits an HTML page, the `validate()` method will be called. In the method, authentication-specific routines are called. At any point in this method, if the authentication has failed, the module must throw a `LoginException`. If desired, the reason for failure can be an argument to the exception. This reason will be logged in the iPlanet Portal Server authentication log.

`init()` should be used if the class has any specific initialization such as loading a JNI library.

`init()` is called once for each instance of the class. Every authentication session creates a new instance of the class. Once a login session is completed the reference to the class is released.

`getUserTokenId()` is called once at the end of a successful authentication session by the iPlanet Portal Server authentication server. This is the string the authenticated user will be known as in the iPlanet Portal Server environment. A login session is deemed successful when all pages in the `.properties` file have been sent and the module has not thrown an exception.

Recommendations

- Add `/opt/SUNWips/classes` to the `CLASSPATH` environment variable
- Naming the authentication module in the following way allows (and forces) the module's class file to be installed with the other auth modules provided with the iPlanet Portal Server software
`com.ipplanet.portalserver.service.auth.module.sample`

About using helpers

A helper is a process that runs separately from the Portal Server but processes the server's authentication requests associated with a given authentication module. In the case of the Portal Server, a helper is a C-language process.

The helper listens for requests on a socket. If data is passed in the clear over the connection, the helper should only accept requests originating from the localhost. The helper can be started at boot time. The startup script is in:

```
/etc/rc3.d/S42ipsserver
```

Some authentication SDKs are supplied as C-language libraries that must be statically linked. If using JNI is undesirable for portability, design, or performance reasons, using an authentication helper is an option.

The UNIX, SafeWord, SecurID, and RADIUS authentication modules employ helpers.

Integrating the Module

After the pluggable authentication module has been written and tested, it must be integrated into the iPlanet Portal Server software with the following procedure.

1. Create a directory for the authentication module under:

```
/opt/SUNWips/lib/
```

2. Put the authentication `.class` file into the directory created for it.

3. Put the corresponding `.properties` file in:

```
/etc/opt/SUNWips/auth/default
```

4. Create an XML file to update the `iwtAuth` component of the Portal Server or domain. See Table 6-4 on page 67 for an example of the XML file.

- a. Make a copy of `/etc/opt/SUNWips/xml/iwtAuth.xml`
For example, `/etc/opt/SUNWips/xml/iwtAuth.xml.update`

- b. Add the values for your module to the new XML file.

Use the example XML file provided in Code Example 6-4 on page 67.

NOTE The new XML file includes only the attributes to be updated. The component name part is omitted.

- c. Import the XML file.

```
# /opt/SUNWips/bin/ipsadmin change component iwtAuth <iwtAuth.xml.update>
```

5. Restart the iPlanet Portal Server.

```
# /opt/SUNWips/bin/ipsserver start
```


Sample Code

The following samples show the form and content of the files associated with an authentication module:

- Sample Properties File
- Sample Login Module Source
- Sample XML File

Sample Properties File

The following sample file `Sample.properties` can be located in:

```
/opt/SUNWips/sample/auth/module
```

where `/opt` is the directory in which it is installed by default.

Code Example 6-2 Sample.properties File

```
SCREEN
TEXT This is a sample login page
TOKEN First Name
TOKEN Last Name

SCREEN
TIMEOUT 30
TEXT Welcome to page 2
PASSWORD Your password

SCREEN
TIMEOUT 60
TEXT Welcome to page 3
TOKEN Enter <REPLACE>'s favorite food
PASSWORD Enter <REPLACE>'s favorite color

SCREEN
TEXT Welcome to page 4
PASSWORD your password
TOKEN anything here
```

Sample Login Module Source

The following sample is in a file named `Sample.java` located in:

`/opt/SUNWips/sample/auth/module/sample`

where `/opt` is the directory in which it is installed by default.

Code Example 6-3 Sample Java Module—`Sample.java`

```
package com.iplanet.portalserver.sample.auth_modules;

import java.util.*;

import com.iplanet.portalserver.auth.server.*;

public class Sample extends Login {

    private String userTokenId;
    private String firstName;
    private String lastName;

    public Sample() throws LoginException{
        System.out.println("Sample()");
    }

    public void init() throws LoginException {
        System.out.println("Sample initialization");
    }

    public void validate() throws LoginException {

        int currentState = getCurrentState();

        if (currentState == 1) {
            firstName = getToken(1);
            lastName = getToken(2);
            if (firstName.equals("") || lastName.equals("")) {
                throw new LoginException("names must not be
empty");
            }
            return;
        }
        else if (currentState == 2) {
            String pass = getToken(1);
            System.out.println("Replace TExt first: " + firstName
+ " last: " + lastName);
            setReplaceText(1, firstName);
            setReplaceText(2, lastName);
            return;
        }
        else if (currentState == 3) {
            String[] tokens = getAllTokens();
            for (int i=0; i<getNumberOfTokens(); i++) {
```

Code Example 6-3 Sample Java Module—Sample.java (*Continued*)

```

        System.out.println("Token-> " + tokens[i]);
    }
    return;
}
else if (currentState == 4) {
    String[] tokens = getAllTokensForState(1);
    for (int i=0; i<getNumberOfTokensForState(1); i++) {
        System.out.println("Token-> " + tokens[i]);
    }
}

    userTokenId = firstName;
}

public String getUserTokenId() {
    return userTokenId;
}

}

```

Sample XML File

The following is a sample XML file. The values in bold type represent the updates made to the XML file. **Replace** the values in **bold type** with the values for the **new** authentication module.

Code Example 6-4 Sample XML File—ispAuth.xml.update

```

<iwt:Att name="iwtAuth-authMenu"
    desc="Authentication Menu"
    type="multichoice"
    idx="a1"
    userConfigurable="TRUE">
    <Val>Radius</Val>
    <Val>SecurID</Val>
    <Val>SafeWord</Val>
    <Val>SKey</Val>
    <Val>Unix</Val>
    <Val>Ldap</Val>
    <Val>NT</Val>
    <Val>Sample</Val>
    <Rperm>ADMIN</Rperm><Rperm>OWNER</Rperm>
    <Wperm>ADMIN</Wperm>
    <CVal>Radius</CVal>
    <CVal>SecurID</CVal>
    <CVal>SafeWord</CVal>
    <CVal>SKey</CVal>
    <CVal>Unix</CVal>

```

Code Example 6-4 Sample XML File—ispAuth.xml.update (Continued)

```

        <CVal>Ldap</CVal>
        <CVal>NT</CVal>
        <CVal>Sample</CVal>
    </iwt:Att>
    <iwt:Att name="iwtAuth-adminAuthModule"
        desc="Admin Authenticator"
        type="singlechoice"
        idx="a4"
        userConfigurable="TRUE">
        <Val>Unix</Val>
        <CVal>Radius</CVal>
        <CVal>Simple</CVal>
        <CVal>SecurID</CVal>
        <CVal>SafeWord</CVal>
        <CVal>SKey</CVal>
        <CVal>Unix</CVal>
        <CVal>Ldap</CVal>
        <CVal>NT</CVal>
        <CVal>Sample</CVal>
        <Rperm>ADMIN</Rperm>
        <Wperm>ADMIN</Wperm>

    </iwt:Att>
    <iwt:Att name="iwtAuth-authenticators"
        desc="Authentication Modules"
        type="multichoice"
        idx=" "
        userConfigurable="TRUE">
        <Val>com.iplanet.portalserver.auth.module.radius.
        Radius</Val>
        <Val>com.iplanet.portalserver.auth.module.securid.
        SecurID</Val>
        <Val>com.iplanet.portalserver.auth.module.safeword.
        SafeWord</Val>
        <Val>com.iplanet.portalserver.auth.module.skey.SKey
        </Val>
        <Val>com.iplanet.portalserver.auth.module.unix.Unix
        </Val>
        <Val>com.iplanet.portalserver.auth.module.ldap.Ldap
        </Val>
        <Val>com.iplanet.portalserver.auth.module.cert.Cert
        </Val>
        <Val>com.iplanet.portalserver.auth.module.nt.NT
        </Val>
        <Val>com.iplanet.portalserver.auth.module.
        application.Application</Val>
        <Val>com.iplanet.portalserver.auth.module.sample.
        Sample</Val>
        <Rperm>ADMIN</Rperm><Rperm>OWNER</Rperm>
        <Wperm>ADMIN</Wperm>
        <CVal>com.iplanet.portalserver.auth.module.radius.
        Radius</CVal>

```

Code Example 6-4 Sample XML File—ispAuth.xml.update (*Continued*)

```
<CVal>com.ipanet.portalserver.auth.module.securid.
SecurID</CVal>
<CVal>com.ipanet.portalserver.auth.module.
safeword.SafeWord</CVal>
<CVal>com.ipanet.portalserver.auth.module.skey.
SKey</CVal>
<CVal>com.ipanet.portalserver.auth.module.unix.
Unix</CVal>
<CVal>com.ipanet.portalserver.auth.module.ldap.
Ldap</CVal>
<CVal>com.ipanet.portalserver.auth.module.cert.
Cert</CVal>
<CVal>com.ipanet.portalserver.auth.module.nt.NT
</CVal>
<CVal>com.ipanet.portalserver.auth.module.
application.Application</CVal>
<CVal>com.ipanet.portalserver.auth.module.
sample.Sample</CVal>
</iwt:Att>
```


Single Signon

Single Signon Overview

The Single Signon provides developers of iPlanet Portal Server API's a mechanism to let users access the applications freely after the initial session signon, rather than prompting for authentication information to access each application during that session. The session/user authentication is established at initial signon by the session server.

At a high-level, single signon application development requires developers to:

- a. Use the Session API to validate an HTTP request from the user into an iPlanet Portal Server session.
- b. Use the Profile and Policy API to access the application-specific authentication information that is stored in the iPlanet Portal Server profile.
- c. Pass that information to the application.

Special Cases

HTTP Basic Authentication is automatically handled by the gateway. It monitors user logins, then writes the URL and encrypted authentication information to the Profile Server.

Similar to HTTP Basic Authentication is *NetFile*. NetFile notes what's been used (username, password, mount information) and remembers it for next time.

A system administrator can also pre populate URLs in the Profile database.

Before logging into the Portal Server the servlet program will print out the value of the session ID.

NOTE	The cookie name would normally be retrieved by the application from the http header.
-------------	--

Instructions for using Single Signon

This section provides information for linking a Single Signon authorization to a user's iPlanet Portal Server desktop.

Command Line Example

iPlanet Portal Server software must be installed to use this sample.

1. Set `IPS_BASE` to the iPlanet Portal Server installation directory.
2. `cd $IPS_BASE/SUNWips/sample/sso`. then type `make`.
3. Copy the class files to the appropriate directory under:

```
$IPS_BASE/SUNWips/lib
```

on the portal server, e.g., the `SSO.class` would be copied to:

```
$IPS_BASE/SUNWips/lib/com/iplanet/portalserver/sso
```

4. Modify the web server configuration.

The web server configuration files are in the directory:

```
$IPS_BASE/netscape/server4/https-servername/config
```

where *servername* is the FQDN of the portal server.

5. Add the following line to the web server `servlets.properties` file:

```
servlet.sso.code=com.iplanet.portalserver.sso.SSO
```

Replace the package and servlet names with the names chosen for this SSO servlet

6. Add the following line to the web server `rules.properties` file:

```
/sso=sso
```

7. Restart the portal server:


```
# etc/init.d/ipsserver start
```

8. Test the servlet by logging in to Portal Server and entering the following URL:

```
https://gateway/http://server:8080/sso
```

Include the iPlanet Portal Server Classes

At a minimum, the Java client application should import the iPlanet Portal Server Profile, Logging, and Session classes, as shown here.

```
package com.iplanet.portalserver.sso;

import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
import com.iplanet.portalserver.session.*;
import com.iplanet.portalserver.profile.*;
import com.iplanet.portalserver.logging.*;
import com.iplanet.portalserver.util.*;
```

Single signon checks to see if the session is valid by looking for the cookie (planted by the Session) with the name iPlanetPortalServer.

Code Example 7-1 SSO.Java

```
public class SSO extends HttpServlet implements SessionListener{

    private Vector v = new Vector();
    private static boolean connectedToMailServer = false;

    public void init(ServletConfig config) throws
    ServletException {
    }
}
```

Code Example 7-1 SSO.Java (Continued)

```

        public void doGet (HttpServletRequest req,
        HttpServletResponse res)
        throws ServletException, IOException {
            v = new Vector();
            SessionID sid = new SessionID(req);
            Session sess=null;
            try {
                sess = Session.getSession(sid);
                v.addElement("sessionID: " + sess.getID());
                v.addElement("userID: " + sess.getClientID());
                v.addElement("domain: " + sess.getClientDomain());

                // Possible states are VALID, INVALID, INACTIVE, DESTROYED
                // we only care whether it is valid or not

                int state = sess.getState(false);
                if (state != Session.VALID) {
                    v.addElement("user session invalid");
                }

                v.addElement("Session: Valid");

                // get the user profile associated with the session

                Profile p = sess.getUserProfile();
                String serverIMAP =
                p.getAttributeString("iwtUser-IMAPServerName");
                String serverSMTP =
                p.getAttributeString("iwtUser-SMTPServerName");
                String userId = p.getAttributeString("iwtUser-IMAPUserId");
                String passWord =
                p.getAttributeString("iwtUser-IMAPPassword");

                if (!connectedToMailServer) {
                    connect(userId, passWord);
                    connectedToMailServer = true;
                }

                v.addElement("IMAP server: " + serverIMAP);
                v.addElement("SMTP server:" + serverSMTP);
                v.addElement("user id: " + userId);
                v.addElement("password:" + passWord);
            }
            catch(SessionException e){
                v.addElement("Session invalid: " + e.getMessage());
            }
            catch(ProfileException e){
                v.addElement("Profile exception: " + e.getMessage());
            }
            catch(Exception e){
                v.addElement("Exception " + e.getMessage());
            }
            send_it(res, v);
        }

```

Code Example 7-1 SSO.Java (*Continued*)

```

    void printProfile(Hashtable h, Profile p, String message) {
    try {
        v.addElement(message);
        for (Enumeration e = h.keys(); e.hasMoreElements();) {
            String s = (String)e.nextElement();
            Enumeration ee = p.getAttribute(s);
            v.addElement(s);
            while (ee.hasMoreElements()) {
                String ss = (String)ee.nextElement();
                v.addElement(ss);
            }
        }
    }
    catch(Exception e){
        v.addElement("Exception " + e.getMessage());
    }
    }

    public void send_it(HttpServletResponse res, Vector st) {
        try {
            ServletOutputStream out = res.getOutputStream();
            res.setContentType("text/html");
            out.println("<HEAD><TITLE> iPS SSO sample
</TITLE></HEAD><BODY>");
            out.println("<h1> SSO iPS Sample </h1>");
            for (int i=0;i<st.size();i++) {
                out.println("<P>" + (String)st.elementAt(i));
            }
            out.println("</BODY>");
            out.close();
        }
        catch (Exception e) {
            o.println("Exception");
        }
    }

    public void sessionChanged(SessionEvent evt) {
        Session sess = evt.getSession();
        try {
            if (sess.getState(false) != Session.VALID) {

            }
        }
    }
    catch (SessionException e) {}

    public String getServletInfo() {
        return "iPS SSO sample";
    }

    private void connect(String user, String pass) {
        // connect to mail server
        return;
    }

```

Code Example 7-1 SSO.Java (*Continued*)

```
    }  
    private static final PrintStream o = System.out;  
}
```

Using the Command Line Interface

Command Line Interface Overview

This chapter describes the command-line interface (`ipsadmin`) available for iPlanet Portal Server administration. Use `ipsadmin` to import XML files to register (or update) iPlanet Portal Server applications or content providers.

How it Works

As iPlanet Portal Server is installed, XML files shipped with the product are imported into the Profile and Policy Server (using `ipsadmin`) to register the existing applications. See `/etc/opt/SUNWips/xml` to see the XML code used.

If new applications are written that should be administered through the iPlanet Portal Server desktop or if expanding on the capabilities of existing modules, it is necessary to write and import an XML file to register the module with the Profile and Policy Server. Additionally, use the `ipsadmin` command to script or automate most routine tasks that could otherwise be accomplished through the Administration Console. For example:

- Create domains
- Create roles
- Add a user

ipsadmin Command

By providing additional or new information to the Profile and Policy Server, the `ipsadmin` command allows the creation or modification of:

- A role
- A user
- A domain
- A component

Usage

```
ipsadmin [-import|-chkxml] xmlfile
```

```
ipsadmin [change] [role|user|domain|component] name [xmlfile]
```

```
ipsadmin [get|delete] [role|user|domain|component] name
```

Where:

`-import` imports the *xmlfile* as a new component in the Profile service.

`-chkxml` checks the validity of the XML and reports errors without making any changes to the Profile. You should use this before importing any data.

xmlfile comprises component name, attributes, and privileges as per the `webtopimport.dtd` file. See Code Example 8-1 for the DTD and annotations.

`create`, `get`, `change`, `delete` are operations that can be performed on a profile.

name is the name of the profile to be operated on.

`role`, `user`, `domain`, `component` are the types of the profile to be operated on.

file is the XML file containing the contents for the operation with regard to attributes and privileges. This uses the `iwt:Att` and `iwtPriv` tags in the `wtimport.dtd` file.

Using ipsadmin

Importing a New Component

1. Create a file `newComponent.xml` which describes what the Profile Server must know about the component:

```
<iwt:Component name="newComponent"
```

Within this section, specify other “newComponent”-wide data, such as description, resource bundle, and index.

```
>
```

```
<iwt:Att name="newComponent-attribute1"
```

Within this section, specify other newComponent-attribute1 related data, such as description, index, default value, and type.

```
>
```

```
</iwt:Att>
```

2. List as many other attributes as you require.

```
<iwt:Priv name="newComponent-privilege1"
```

Within this section, specify other newComponent-privilege1 related data, such as description, index, default value, or type.

```
>
```

```
</iwt:Att>
```

```
</iwt:Component>
```

3. Issue the ipsadmin command:

```
# ipsadmin -import newComponent.xml
```

NOTE ipsadmin registers the attributes and privileges in the iPlanet Portal Server LDAP data store, but does not remove this metadata if the component is deleted later with the `ipsadmin delete` command. A subsequent addition of the same component (for example, with extra attributes) will generate warnings that the attributes and privileges are already registered; these are harmless errors and are for information only.

Creating a New Domain

1. Create an XML file which contains attributes and privileges for this Domain.

This XML file looks like the XML in the “Importing a New Component” example, except that it does not have the enclosing `<iwt:Component>` tags and it could have attributes and privileges from any of the components currently imported. It contains `<iwt:Att` and `<iwt:Priv` tags only.

2. Issue the `ipsadmin` command:

```
# ipsadmin create domain SampleDomain SampleDomain.xml
```

Creating a New Role

1. Create an XML file which contains attributes and privileges for this role (Employee, in this example).

This XML file looks like the XML in the “Importing a New Component” example, except that it does not have the enclosing `<iwt:Component>` tags and it could have attributes and privileges from any of the components currently imported. It contains `<iwt:Att` and `<iwt:Priv` tags only.

2. Issue the `ipsadmin` command:

```
# ipsadmin create role /SampleDomain/Employee xmlfile
```


Creating a New User and Assigning a Role

1. Create an XML file which contains attributes and privileges for this user.

This XML file looks like the XML in the “Importing a New Component” example, except that it does not have the enclosing `<iwt:Component>` tags and it could have attributes and privileges from any of the components currently imported. It contains `<iwt:Att` and `<iwt:Priv` tags, plus these tags:

```
<iwt:Att name="iwtUser-role" >
<Val>/SampleDomain/Employee</Val>
</iwt:Att>
```

2. Execute the `ipsadmin` command:

```
# ipsadmin create user SampleDomain/decoy xmlfile
```

Note that if the role attribute is omitted, `ipsadmin` will create the user, but issue a warning that the role is not set. If that happens, the role can always be added later using the `ipsadmin change` command.

Reading (Getting) a Profile

Use `ipsadmin` to read information out of the Profile Server, to more easily update specific information or to modify the extracted information to create a similar, new entry.

```
# ipsadmin get domain SampleDomain
# ipsadmin get component newComponent
# ipsadmin get role /SampleDomain/Employee
# ipsadmin get user /SampleDomain/decoy
```

All display the attributes and privileges on stdout via the `<iwt:Att` and `<iwt:Priv` tags. The output can be saved in a file and later used for updating or creating another profile.

Changing a Profile

Use `ipsadmin` to modify existing Profile entries by specifying change on the command line as well as the type of component that the XML specifies to change.

Here is an example on how to modify a user's first name and last name, assume the user is `user1` under domain `dom1`.

1. Create an XML file named `/tmp/user.xml` which contains following tags:

```
<iwt:Att name="iwtUserInfoProvider-firstName">
<Val>FirstName</Val>
</iwt:Att>
<iwt:Att name="iwtUserInfoProvider-lastName">
<Val>LastName</Val>
</iwt:Att>
```

The attributes name for user's first name and last name are:

- o "iwtUserInfoProvider-firstName"
- o "iwtUserInfoProvider-lastName"

All the iPlanet Portal Server defined attributes and privileges could be find in the XML files under `/etc/opt/SUNWips/xml` directory.

2. Issue the `ipsadmin` command

```
# ipsadmin change user /dom1/user1 /tmp/user.xml
```

The XML specifies what needs to be changed and values enclosed within the `iwt:Att` and `iwt:Priv` tags.

Deleting a Profile

Use `ipsadmin` to delete existing Profile entries by specifying `delete` on the command line. For example:

```
# ipsadmin delete user /SampleDomain/decoy
# ipsadmin delete role /SUN/Employee
# ipsadmin delete domain SUN
# ipsadmin delete component newComponent
```

Sample Code

Code Example 8-1 XML Sample Compliant with `import.dtd`

```
<!-- Component : -->
<!--     name:name of the component -->
<!--     ver :Version no of this DTD -->
<!--     desc:Brief description in <3 words -->
<!--     resB:Relative path of resource bundle to -->
<!--           use for getting l18n desc version -->
<!--     idx :Index into res bundle to get l18n -->
<!--           version of desc. -->
<!--     Att*,Priv* : privileges and attributes -->
<!-- Naming Convention : -->
<!--     Component name : [a-zA-Z][a-zA-Z0-9]* -->
<!--     Attribute name : <ComponentName>-[[a-zA-Z0-9-]]+ -->
<!--     Privilege name : <ComponentName>-[[a-zA-Z0-9-]]+ -->
<!--     Absolutely NOT allowed : [./#_] -->
<!-- Special/Reserved attribute names: -->
<!--     desc stored as attribute called -->
<!--           "Description" (string) -->
<!--     idx stored as attribute called -->
<!--           "DescIndex" (string) -->
<!--     resB stored as attribute called -->
<!--           "ResourceBundle" (string) -->
<!-- All XML ATTs tagged with #IMPLIED are optional, -->
<!-- All XML ATTs tagged with #REQUIRED are mandatory -->

<ELEMENT iwt:Component (iwt:Att*, iwt:Priv*)>
<!ATTLIST iwt:Component
    name CDATA #REQUIRED
```

Code Example 8-1 XML Sample Compliant with import.dtd (Continued)

```

<!-- Component : -->
    ver CDATA #FIXED "1.0"
    desc CDATA #IMPLIED
    resB CDATA #IMPLIED
    idx CDATA #IMPLIED>

<!-- Privilege : -->
<!--     name:Name of the privilege -->
<!--     type:Type of the privilege -->
<!--     desc:Brief description in <3 words -->
<!--     idx :Index into resB to find l18n version -->
<!--           of the description -->
<!--     val :Default value for boolean type priv : -->
<!--           true=ALLOW, false=DENY -->
<!--     Dlst:Deny List for list type privileges -->
<!--     Alst:Allow List for list type privileges -->
<!--     Special privilege : -->
<!--           "Execute" (boolean) represents execute -->
<!--           permission -->

<!ELEMENT iwt:Priv (iwt:Dlst*,iwt:Alst*)>
<!ATTLIST iwt:Priv
    name CDATA #REQUIRED
    type (boolean|list) "boolean"
    desc CDATA #IMPLIED
    idx CDATA #IMPLIED
    val (true|false) #IMPLIED
>

<!-- Attribute -->
<!--     name:Name of the Attribute -->
<!--     desc:Brief description in <3 words -->
<!--     idx :Index into resB to find l18n version -->
<!--     userConfigurable:flag to indicate if attribute -->
<!--           value is allowed to be specified on a -->
<!--           per User/Role basis -->
<!--     type:Datatype of the value -->
<!--           of the description -->
<!--     Val:Value of the attribute - multiple for-->
<!--           "list" type attributes. -->
<!--     RPerm:List of roles allowed to read the value -->
<!--           Special keywords : ADMIN : admin role -->
<!--                               OWNER : allow owner -->
<!--     WPerm:List of roles allowed to write the value -->
<!--           Special keywords : ADMIN : admin role -->
<!--                               OWNER : allow owner -->
<!--     CVal:Possible choice values for choice* type -->
<!--           attributes. -->

<!ELEMENT iwt:Att (iwt:Val*,iwt:RPerm*,iwt:WPerm*,iwt:CVal*)>
<!ATTLIST iwt:Att
    name CDATA #REQUIRED
    desc CDATA #IMPLIED
    idx CDATA #IMPLIED
    userConfigurable (TRUE|FALSE) "TRUE"

```

Code Example 8-1 XML Sample Compliant with import.dtd (*Continued*)

```

<!-- Component :                                -->
      type (string | number | boolean | singlechoice |
multichoice
      |protected | stringlist | numberlist | binary )
"string"
>
<!ELEMENT iwt:Val (#PCDATA)*>
<!ELEMENT iwt:Dlst (#PCDATA)*>
<!ELEMENT iwt:Alst (#PCDATA)*>
<!ELEMENT iwt:CVal (#PCDATA)*>
<!ELEMENT iwt:Rperm (#PCDATA)*>
<!ELEMENT iwt:Wperm (#PCDATA)*>

<!-- Example                                -->

```


Using the iPlanet Portal Server APIs

Instructions for using the HelloServlet

1. Set `IPS_BASE` to the iPlanet Portal Server installation directory.
2. `cd $IPS_BASE/SUNWips/sample/api` then type `make`.
3. Copy the class files to the appropriate directory on the Portal Server under:

```
$IPS_BASE/SUNWips/lib
```

For example, all class files would be copied to:

```
$IPS_BASE/SUNWips/lib/com/iplanet/portalserver/api
```

4. Modify the web server configuration.

The web server configuration files are in the directory:

```
$IPS_BASE/netstage/server4/https-servername/config
```

where *servername* is the FQDN of the portal server.

5. Add the following line to the web server `servlets.properties` file:

```
servlet.helloservlet.code=com.ipplanet.portalserver.api.HelloServlet
```

6. Replace the package and servlet names with the names you have chosen for this HelloServlet

7. Add the following line to the web server `rules.properties` file:

```
/helloservlet=helloservlet
```

8. Import `iwtHelloServlet.xml` as root using `ipsadmin`:

```
$IPS_BASE/SUNWips/bin/ipsadmin -import iwHelloServlet.xml
```

9. Copy file `iwtHelloServlet.properties` to `$IPS_BASE/SUNWips/locale` directory
10. Restart the portal server:

```
# etc/init.d/ipsserver start
```

11. Test the servlet by logging in to Portal Server and entering the following URL:

```
https://gateway/http://server:8080/HelloServlet
```

HelloServlet Properties

Code Example 9-1 HelloServlet.properties

```
a1=Hello Application Profile
a2=Any user friendly name
a3=Your favourite color
a4=Hello Application
a5=Hello Application execute privilege
a6=Hello Application change color privilege
```

HelloServlet XML

Code Example 9-2 HelloServlet.XML

```
<iwt:Component name="HelloServlet"
  ver="1.0"
  desc="Hello Application Profile"
  resB="HelloServlet"
  idx="a1">

<!-- String Attribute -->
<iwt:Att name="HelloServlet-name"
  desc="Any user friendly name "
  type="string"
```


Code Example 9-2 HelloServlet.XML (*Continued*)

```

        idx="a2"
        userConfigurable="TRUE">
        <Val></Val>
        <Rperm>ADMIN</Rperm><Rperm>OWNER</Rperm>
        <Wperm>ADMIN</Wperm><Wperm>OWNER</Wperm>
    </iwt:Att>

    <!-- String Attribute -->
    <iwt:Att name="HelloServlet-color"
        desc="Your favourite color"
        type="string"
        idx="a3"
        userConfigurable="TRUE">
        <Val></Val>
        <Rperm>ADMIN</Rperm><Rperm>OWNER</Rperm>
        <Wperm>ADMIN</Wperm><Wperm>OWNER</Wperm>
    </iwt:Att>

    <!-- String Attribute, Global/Platform attribute -->
    <!-- Hence userconfigurable is set to false -->
    <iwt:Att name="HelloServlet-title"
        desc="Application description"
        type="string"
        idx="a4"
        userConfigurable="FALSE">
        <Val>HelloServlet</Val>
        <Rperm>ADMIN</Rperm><Rperm>OWNER</Rperm>
        <Wperm>ADMIN</Wperm><Wperm>OWNER</Wperm>
    </iwt:Att>

    <!-- Boolean privilege -->
    <iwt:Priv name="HelloServlet-execute"
        type="boolean"
        desc="Hello Application execute privilege"
        userConfigurable="TRUE"
        idx="a5"
        val="true">
        <Rperm>ADMIN</Rperm><Rperm>OWNER</Rperm>
        <Wperm>ADMIN</Wperm>
    </iwt:Priv>

    <!-- List type privilege -->
    <iwt:Priv name="HelloServlet-changeColor"
        type="list"
        desc="Hello Application change color privilege"
        userConfigurable="TRUE"
        idx="a6">

        <Rperm>ADMIN</Rperm><Rperm>OWNER</Rperm>
        <Wperm>ADMIN</Wperm>
    </iwt:Priv>
</iwt:Component>

```

Prints HTML Output

Code Example 9-3 HTML Output

```

    private void printMessage(HttpServletResponse res)
    throws Exception {
        res.setContentType("text/html");
        ServletOutputStream out = res.getOutputStream();
        out.println("<html>");
        out.println("<head><title>HelloApp</title></head>");
        out.println("<body bgcolor="+ color);
        out.println("<BR>");
        out.println("<CENTER>");
        out.println("<h2>");
        // Check if User is allowed to execute
        // the Hello Application
        if (UserIsAllowed) {
            out.println("Hello User: "+ name+ " !!");
        }
        if (!changeColorIsAllowed) {
            out.println("You are not allowed to change color.");
        }
        else {
            out.println("Sorry "+ name+ " !!");
            out.println("You are not allowed to execute the Hello
application.");
        }
        out.println("</h2>");
        out.println("</CENTER>");
        out.println("</body></html>");
    }
}

```

Before logging into the Portal Server the servlet program will print out the value of the session ID.

NOTE The cookie name would normally be retrieved by the application from the http header.

Setting Privileges

Attributes and Privileges

This servlet is a sample which uses 3 main APIs (Session API, Profile API, Logging API)

Code Example 9-4 Attributes and Privileges

```

public class HelloServlet extends HttpServlet {

    // Attributes and Privileges as defined in the XML file
    private static final String logfile = "HelloApp.log";
    private static final String COLOR_ATTR = "HelloServlet-color";
    private static final String NAME_ATTR = "HelloServlet-name";
    private static final String EXEC_PRIV= "HelloServlet-execute";
    private static final String CHANGECOLOR_PRIV=
"HelloServlet-changeColor";

    // Local variables
    private LogManager lmgr= null;
    private LogRecord log= null;
    private String color= "#CCCCFF";
    private String name= null;
    private boolean UserIsAllowed= false;
    private boolean changeColorIsAllowed= false;

```

Initializing the Servlet

Code Example 9-5 Initialize the Servlet

```

public void init(ServletConfig config) throws ServletException {
    super.init(config);
}

```

Session API Examples

HTTP Request and Response

Parameter requests the HTTP request, parameter responds the HTTP response. If an I/O error has occurred, an I/O exception will be thrown.

Code Example 9-6 HTTP Request and Response

```

public void doGet(HttpServletRequest req, HttpServletResponse res)
    try {

        // Get a Session object
        Session sess = getSession(req);

        // Get user profile name

```

Code Example 9-6 HTTP Request and Response (*Continued*)

```

        Profile p = getProfileName(sess);

        // Get the logManager and start logging
        lmgr = startLogging(sess);

        // Get AccessControl information
        getAccessControlInfo(sess, p);

        // Print the output
        printMessage(res);

    } catch (LogException e) {
        log = new LogRecord("Logging: ", "Error in Logging");
    } catch (SessionException e) {
        log = new LogRecord("Session: ", "Error in Session");
    } catch (ProfileException e) {
        log = new LogRecord("Profile: ", "Error in Profile");
    } catch (Exception e) {
        log = new LogRecord("Error: ", " Printing Hello");
    } finally {
        if (log == null)
            log = new LogRecord("Hello Application:", "Success");

        try {
            lmgr.write(log, logfile);
        } catch (Exception le) {
            System.out.println("Error: "+le);

```

Session Event

If the user logs out of his session or if user exceeds maximum idle time then the Hello application returns.

Code Example 9-7 SessionEvent

```

class HelloSessionListener implements SessionListener {

    public void sessionChanged(SessionEvent e) {
        Session sessionEvt = null;

        // if the session is still valid, just return
        // without doing anything
        try {
            sessionEvt = e.getSession();
            if (sessionEvt.getState(false) == Session.VALID)
                return;
            else {
                // clean up profile before quitting
            }

```

Code Example 9-7 SessionEvent (Continued)

```
class HelloSessionListener implements SessionListener {
    } catch (Exception se) {}
}
}
```

Get a Session

Method handles Session and gets the user Session object and adds a SessionListener.

Code Example 9-8 GetSession

```
SessionID sid = new SessionID(req);
Session sess = Session.getSession(sid);
    if (sessionEvt.getState(false) == Session.VALID)
// sessionChanged() is called if a SessionEvent occurs
sess.addSessionListener(new HelloSessionListener());
}
```

Profile API Examples

Modify an Attribute

Modify an attribute for the Hello application and test if ProfileChanged is called.

Code Example 9-9 Modify an Attribute

```
public class HelloProfileListener implements ProfileListener {
    public void profileChanged(ProfileEvent notify){
        Profile p = notify.getProfile();
        int type = notify.getType();

        // Either the color or the name attribute may have changed
        // Get the new values for these attributes.
        if (type == ProfileEvent.PROFILE_CHANGE) {
            try {
                color = p.getAttributeString(COLOR_ATTR);
                name = p.getAttributeString(NAME_ATTR);
            } catch (ProfileException pe) {
                System.out.println("Profile: getAttribute() failed");
            }
        }
    }
}
```

Code Example 9-9 Modify an Attribute (*Continued*)

```

        }
        return;
    } else {
        // no attributes were changed
        // profiles were created or deleted
        return;
    }
}
}

```

Get User Profile

Method handles user Profile and gets the user profile name and adds a profile listener for any attribute changes in the current user profile.

Code Example 9-10 GetProfileName

```

private Profile getProfileName(Session s)
    throws ProfileException {
    Profile p = s.getUserProfile();
    this.name = p.getProfileName();

    // profileChanged() is called if a ProfileEvent occurs
    p.addProfileListener(new HelloProfileListener());
    return p;
}

```

Policy Checking

Method handles user policy, checks if the user is allowed to execute the hello application.

Code Example 9-11 Policy Checking

```

private void getAccessControlInfo(Session s, Profile p)
    throws ProfileException {
    if (p.isAllowed(EXEC_PRIV)) {
        this.UserIsAllowed = true;
    }

    // The CHANGECOLOR_PRIV is defined as a list type privilege.
    // What's in the list is domains. If the user domain is in the
    // privilege CHANGECOLOR_PRIV's allow list, the user is allowed
    // to change color. Otherwise, the user is denied to change
    color.
}

```

Code Example 9-11 Policy Checking (*Continued*)

```
private void getAccessControlInfo(Session s, Profile p)
if (p.isAllowed(CHANGECOLOR_PRIV, s.getClientDomain(),
Profile.REGULAR)) {
    this.changeColorIsAllowed = true;
}
}
```

Log API Example

Method handles Logging

Application creates a log file "Hello.log" and logs the first log entry.

Code Example 9-12 Create Log File

```
private LogManager startLogging(Session s)
    throws LogException {
    LogManager lm = new LogManager(s);
    lm.create(logfile);

    return lm;
}
```


HTML Templates

Setting up Login Pages for Different Domains

HTML templates can be edited to make substantive changes to layout or design of pages, or to add extra functionality, beyond the services possible through the Administration Console.

NOTE	Strong HTML skills as well as a thorough understanding of web servers, and server-side includes, are required to edit the template files. If a template file is corrupted, it may be necessary to restore the original files from the iPlanet Portal Server CD-ROM to recover and gain access to the system.
-------------	--

How Authentication Templates Work

HTML template files control the layout and source of the iPlanet Portal Server Desktop and of the other screens that users see. The templates are located on the Portal Server in the directory:

```
/etc/opt/SUNWips/auth/default
```

Templates for Customizing the Authentication Pages

These templates allow customizing the login, logout, and time-out screens.

In the `/etc/opt/SUNWips/auth/default` directory, there are `.html` files that control the overall appearance and `.properties` files that control the sequence of prompts and the exchange of information between the user and the authentication module.

NOTE See the Chapter 6, “Pluggable Authentication API” for more information about `<authentication-module>.properties` files.

The login pages come from a set of template HTML files. The default set of these is located at `/etc/opt/SUNWips/auth/default`

To customize a login for different domains:

1. Go to the server machine (do the same to all server machines if there are multiple servers).
2. Create a directory named the same as the name of the domain.
3. Copy all the `.properties`, `.html` and `.gif` files into that directory.
4. Customize the files in that directory for that domain.

Any domain that does not have it's own directory of templates will use the default set in:

```
/etc/opt/SUNWips/auth/default
```

For example, if there are three domains: `dm1`, `dm2`, `dm3`, and the login will be customized for `dm1`, the directories will look like this:

```
/etc/opt/SUNWips/auth/default
```

```
/etc/opt/SUNWips/auth/dm1
```

Both would contain a full set of the `properties/html/gif` files. The login to `dm1` would use the set in `/etc/opt/SUNWips/auth/dm1/`, and the other domains would use the default set in `/etc/opt/SUNWips/auth/`

The various files are:

Table 10-1 HTML Template Files

File Name	Description
<code>login_menu.html</code>	Is sent when more than one authentication module is configured. This gives the iPlanet Portal Server end user a choice of which module to use for authentication. The text <code><subst data="rows">No menu?</subst></code> <i>must</i> be somewhere in the document. It generates a list of URLs to the authentication modules.

Table 10-1 HTML Template Files (*Continued*)

File Name (<i>Continued</i>)	Description (<i>Continued</i>)
login_fail_template.html	Is sent when authentication has failed. This page contains no required sections.
login_license_fail.html	Is sent when there are no more licenses. This page contains no required sections.
login_reauth_menu.html	Is sent when an iPlanet Portal Server end user's session has been inactive for the time set in the Administration Console. It contains a link for re authentication. Do not change the JavaScript in this page.
login_trustProxy_warning.html	Is presented to iPlanet Portal Server end users who log in to the iPlanet Portal Server Desktop using Netscape and do not have the browser configured to accept all cookies. In this case the end user cannot start the iPlanet Portal Server Java applets. If this page is not to be displayed because the end user is not allowed to run Java applets, replace the contents of this page with an HTML refresh tag that contains zero time-out and is redirected to /login/default.
login_template.html	Is sent for individual authentication modules such as RADIUS or UNIX. The seven subset text segments must remain after modification. This page is also sent when logging in to the iPlanet Portal Server Administration Console.
logout.html	Is called after the iPlanet Portal Server end user selects the logout link on the iPlanet Portal Server Desktop. It contains no required sections.
login_timeout_template.html	Is called during an authentication session if the iPlanet Portal Server end user does not submit the login form within the specified time. It has no required sections.
login_reauth_admin.html	Is sent when the administration session has expired. It contains a link for re authentication.
login_timeout_admin.html	Is called during an administration session if nothing is entered within the specified time. It has no required text.
login_fail_admin.html	Is sent when authentication has failed. This page contains no required text.
logout_admin.html	Is called after the iPlanet Portal Server administrator selects the logout link on the iPlanet Portal Server Administration Console. It contains no required sections.

How Desktop Templates Work

HTML template files control the layout and source of the iPlanet Portal Server Desktop and of the other screens that users see. The templates are located on the Portal Server in the directory `/etc/opt/SUNWips/desktop`.

Templates for Customizing the iPlanet Portal Server Desktop

HTML template files control the layout and source of the iPlanet Portal Server Desktop. The templates are:

<code>advancedTemplate.html</code>	Controls the Advanced page in the end user's iPlanet Portal Server Desktop. By default, it is set to allow an applet that downloads NetMail and installs it on a local client, but you can add other functionality of your choice.
<code>feedbackTemplate.html</code>	Controls the appearance and layout of the feedback form in the end user's iPlanet Portal Server Desktop.
<code>prefTemplate.html</code>	Controls the appearance and layout of the Edit Preferences page in the end user's iPlanet Portal Server Desktop.
<code>userTemplate.html</code>	Controls the Front Page in the end user's iPlanet Portal Server Desktop.

These files can be edited to change the appearance, the information presented, and the links. Add or remove various features according to the corporate policy or security policy. These files can also be copied to build new templates.

The following tag definitions are used in the Desktop Pages.

Tag Definition	Description
<code>noCache</code>	Various HTML directives to try and stop browser from caching pages.
<code>productName</code>	Replaced with Platform-productName attr.
<code>openFrontPage</code>	js code to return to front page.

Tag Definition (<i>Continued</i>)	Description (<i>Continued</i>)
<code>style</code>	Various style/class defs used by the templates (HTML).
<code>menubar</code>	HTML for the black menubar on the dt pages.
<code>banner</code>	The desktop's banner (HTML).
<code>title</code>	On dt's edit provider page, is <code>iwt<provider>-title</code> attr.
<code>providerName</code>	Providers <code>iwt<provider>-name</code> attr.
<code>inlineError</code>	Place holder to insert an inline error msg in dt pages (blank if no error).
<code>contentOptions</code>	The content of the provider edit page that is generated by the provider.
<code>errMessage</code>	The actual text of the error message, inserted into the inline error template.
<code>thinProviders</code>	HTML for the "thin" providers.
<code>wideProviders</code>	HTML for the "wide" providers.
<code>size</code>	Width for provider columns.
<code>provider_cmds</code>	The menubar for the provider (the buttons).
<code>content</code>	Content for the provider.
<code>providerTitle</code>	<code>iwt<provider>-title</code> attr.
<code>openURLInParent</code>	js code to open a url.
<code>popupMenubar</code>	Menubar for the detached provider.
<code>providerContent</code>	Provider content in popup form.
<code>arrangeProvider</code>	js code.
<code>removeProvider</code>	js code to remove a provider.
<code>performSub*</code>	js code.
<code>selectAll</code>	js code to select all.
<code>switchColumns</code>	js code.
<code><left center right>UserProviderList</code>	List boxes of providers.
<code>launchPopup</code>	js code to detach provider.

Tag Definition (<i>Continued</i>)	Description (<i>Continued</i>)
serviceTimeout	iwtDesktop-serviceTimeout attr.
layout<n>Checked	Indicates which layout option is checked.
help_link	Help link for a page.
resourceCount, resourceList, windwoOptions	Used dynamically by some edit pages.
bookmarks	The content for the bookmark provider iwtXXXXXX (in ui edit template); get subbed for that attr.
timezoneList	HTML list element.
timezones targetCount	Number of targets in netlet provider.
newRuleSelect	Indicator of new rule bing added in netlet provider.
targetList	List of targets for NP.
content	Content for the netlet provider.
apps	Content for the app provider.

HTTP/XML Interface

HTTP/XML Interface Overview

Programs or applications written in a non-Java language can also exchange information with the iPlanet Portal Server product, although non-Java-based clients are considerably less efficient to use and are more cumbersome to implement. This appendix details the processes and issues involved in communicating with the Portal Server using the exposed HTTP/XML interface (or API, of sorts).

In general, this appendix applies to writing a standalone non-Java-based application that is authenticated through the session ID from an iPlanet Portal Server session or an application that will use any of the other exposed interfaces, including Profile and Policy services, Logging, Session, or other interfaces. Non-Java applications must explicitly exchange information with the server using an over-the-wire protocol, rather than simply passing objects from class to class as a Java-based client implementation can.

The mechanism for passing information from client application to iPlanet Portal Server and back is through *http posts*. The structure of the information to pass is defined by XML-based DTDs for each service.

Therefore, a minimal non-Java-based iPlanet Portal Server client application must implement:

- HTTP 1.1 compliant client implementation to send information to the iPlanet Portal Server server
- HTTP 1.1 compliant server implementation to receive information from the iPlanet Portal Server if asynchronous communication or iPlanet Portal Server-initiated communication is required
- XML parser to interpret the data stream coming from the iPlanet Portal Server software

- A means of producing valid XML that is compliant with a given DTD

Exchanging Information Between the Client and the Server

The conversation between client and server is as follows:

1. Get the property "ips.naming.url" from /etc/opt/SUNWips/platform.conf as the naming URL.
2. Client posts a name request to the naming URL.
3. Server returns name response to client.
4. Client parses name response to determine URLs for Session service, Profile Service, and Logging Service.
5. Client posts a session request to the Session Service URL.
6. Server returns session response to client.
7. Client parses session response to determine session ID to use.
8. Client posts a profile request to the Profile Service URL with the session ID obtained in Step 6.
9. Server returns profile response to client.
10. Client parses profile response to obtain profile attributes.
11. Client posts a logging request to the Logging Service URL.
12. Server returns logging response to client.
13. Client parses logging response to decide whether the logging request succeeded.

XML DTDs

The following sections outline the DTDs that define information structures for initial data exchanges between client software and the Portal Server. These exchanges negotiate the URL for the subsequent exchanges with a specific application (registered with the Profile and Policy server).

PLL Request Set DTD

The request from the Portal Server to a name response will comply with the following DTD.

Code Example A-1 PLL RequestSet DTD

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- This DTD is used by PLL -->
<!DOCTYPE RequestSet [
<!ELEMENT RequestSet (Request)+>
<!ATTLIST RequestSet
    vers CDATA #REQUIRED
    svcid CDATA #REQUIRED
    reqid CDATA #REQUIRED>
<!ELEMENT Request (#PCDATA)*>
<!ATTLIST Request
    dtdid CDATA #IMPLIED>
]>
```

PLL Response Set DTD

The response from the Portal Server to a name request will comply with the following DTD.

Code Example A-2 PLL ResponseSet DTD

```
<?xml version="1.0">
<!-- This DTD is used by PLL -->
<!DOCTYPE ResponseSet [
<!ELEMENT ResponseSet (Response)+>
<!ATTLIST ResponseSet
    vers CDATA #REQUIRED
    svcid CDATA #REQUIRED
    reqid CDATA #REQUIRED>
<!ELEMENT Response (#PCDATA)*>
<!ATTLIST Response
    dtdid CDATA #IMPLIED>
]>
```

PLL Notification Set DTD

The response from the Portal Server to a notification will comply with the following DTD.

Code Example A-3 PLL NotificationSet DTD

```
<?xml version="1.0">
<!-- This DTD is used by PLL -->
<!DOCTYPE NotificationSet [
<!ELEMENT NotificationSet(Notification)+>
<!ATTLIST NotificationSet
    vers    CDATA #REQUIRED
    svcid   CDATA #REQUIRED
    notid    CDATA #REQUIRED>
<!ELEMENT Notification(#PCDATA)*>
<!ATTLIST Notification
    dtdid   CDATA #IMPLIED>
]>
```

Naming Response DTD

The request from the Portal Server to a name response will comply with the following DTD.

Code Example A-4 NamingResponse DTD

```
<?xml version="1.0">
<!DOCTYPE NamingResponse [
<!ELEMENT NamingResponse (GetNamingProfile)
<!ATTLIST NamingResponse
    vers    CDATA #REQUIRED
    reqid   CDATA #REQUIRED>
<!ELEMENT GetNamingProfile (Attribute*|Exception)>
<!ELEMENT Attribute EMPTY>
<!ATTLIST Attribute
    name    CDATA #REQUIRED
    value   CDATA #REQUIRED>
<!ELEMENT Exception (#PCDATA)>
]>
```

Naming Request DTD

The initial request from a client application to the Portal Server always goes to the URL:

`http://server-name:port/NamingService`

with a post, and must be XML compliant with the following DTD.

Code Example A-5 NamingRequest DTD

```
<?xml version="1.0">
<!DOCTYPE NamingRequest [
<!ELEMENT NamingRequest (GetNamingProfile)>
<!ATTLIST NamingRequest
    vers CDATA #REQUIRED
    reqid CDATA #REQUIRED>
<!ELEMENT GetNamingProfile EMPTY>
]>
```

Naming Request XML

Code Example A-6 NamingRequest XML

```
<?xml version="1.0" standalone="yes"?>
<RequestSet vers="1.0" svcid="ips.naming" reqid="0">
<Request>
// BEGIN service specific XML message
<![CDATA[<NamingRequest vers="1.0" reqid="0">
<GetNamingProfile>
</GetNamingProfile>
</NamingRequest>]]>
// END service specific XML message
</Request>
</RequestSet>
```

Naming Response XML

Code Example A-7 NamingResponse XML

```
<ResponseSet vers="1.0" svcid="ips.naming" reqid="0">
<Response>

  // BEGIN service specific XML
  <![CDATA[<NamingResponse vers="1.0" reqid="0">
<GetNamingProfile>
<Attribute name="iwtPlatform-servers"
value="huanghe.red.iplanet.com "></Attribute>
<Attribute name="iwtNaming-loggingClass"
value="com.iplanet.portalserver.logging.service.LogService"></At
tribute>
<Attribute name="iwtNaming-resourceBundle"
value="iwtNaming"></Attribute>
<Attribute name="iwtNaming-sessionClass"
value="com.iplanet.portalserver.session.service.SessionRequestHa
ndler"></Attribute>
<Attribute name="iwtNaming-profileClass"
value="com.iplanet.portalserver.profile.service.ProfileService">
</Attribute>
<Attribute name="iwtNaming-description"
value="Naming"></Attribute>
<Attribute name="iwtNaming-profileURL"
value="http://huanghe.red.iplanet.com:8080/profileservice"></Att
ribute>
<Attribute name="iwtNaming-loggingURL"
value="http://huanghe.red.iplanet.com:8080/loggingservice"></Att
ribute>
<Attribute name="iwtNaming-sessionURL"
value="http://%host:8080/sessionservice"></Attribute>
</GetNamingProfile></NamingResponse>]]>

  // END service specific XML

</Response>
</ResponseSet>
```

Further exchanges are determined by the specific API and DTDs for the service being addressed.

Session-Related DTD and XML

Implementing the Session API functionality in non-Java languages requires providing and parsing XML-based data streams to communicate with the logging processes on the server. The following sections detail the DTDs that describe acceptable communication with the Session processes.

NOTE If accessing Session API functionality through Java programs, it is not necessary to be aware of these implementation specifics, because the Java interface makes these transparent to Java programmers. See the first section of this chapter to learn to access Session API functionality through Java programs.

SessionNotification DTD

The Session Notification DTD establishes for structure for the server to use when providing information about the session status to a requesting client.

Code Example A-8 NotificationSet DTD

```
<?xml version="1.0">
<!-- This DTD is used by PLL -->
<!DOCTYPE NotificationSet [
<!ELEMENT NotificationSet(Notification)+>
<!ATTLIST NotificationSet
    vers CDATA #REQUIRED
    svcid CDATA #REQUIRED
    notid CDATA #REQUIRED>
<!ELEMENT Notification(#PCDATA)*>
<!ATTLIST Notification
    dtdid CDATA #IMPLIED>
]>
```

Session Request DTD

The Session Request DTD establishes the information structure for a client application to modify or get a session from the Session server.

Code Example A-9 SessionRequest DTD

```
<?xml version="1.0">
<!DOCTYPE SessionRequest [
  <!ELEMENT SessionRequest (GetSession |
                           GetValidSessions |
                           DestroySession |
                           Logout |
                           AddSessionListener |
                           AddSessionListenerOnAllSessions |
                           SetProperty)>

  <!-- The reset attribute indicates whether resets the latest
  access time -->
  <!-- ATTTLIST SessionRequest
  vers    CDATA #REQUIRED
  reqid   CDATA #REQUIRED>
  <!ELEMENT GetSession (SessionID)>
  <!-- ATTTLIST GetSession
  <!-- The reset attribute indicates whether resets the latest
  access time -->
  <!-- reset    CDATA #REQUIRED>
  <!ELEMENT GetValidSessions (SessionID)>
  <!-- ELEMENT DestroySession (SessionID, DestroySessionID)>
  <!-- ELEMENT Logout (SessionID)>
  <!-- ELEMENT AddSessionListener (SessionID, URL)>
  <!-- ELEMENT AddSessionListenerOnAllSessions (SessionID, URL)>
  <!-- ELEMENT SetProperty (SessionID, Property)>
  <!-- ELEMENT Property>
  <!-- ATTTLIST Property
  name    CDATA #REQUIRED
  value   CDATA #REQUIRED>
  <!-- ELEMENT SessionID (#PCDATA)>
  <!-- ELEMENT DestroySessionID (#PCDATA)>
  <!-- ELEMENT URL (#PCDATA)>
]>
```

Session Response DTD

The Session Response DTD defines the information structure for the server's response to a session request.

Code Example A-10 SessionResponse DTD

```
<?xml version="1.0">
<!DOCTYPE SessionResponse [
<!ELEMENT SessionResponse(GetSession |
                           GetActiveSessions |
                           DestroySession |
                           Logout |
                           AddSessionListener |
                           AddSessionListenerOnAllSessions |
                           SetProperty)>

<!--ATTLIST SessionResponse
      vers      CDATA #REQUIRED
      reqid     CDATA #REQUIRED-->
<!ELEMENT GetSession (Session|Exception)>
<!ELEMENT GetActiveSessions (SessionList|Exception)>
<!ELEMENT DestroySession (OK|Exception)>
<!ELEMENT Logout (OK|Exception)>
<!ELEMENT AddSessionListener (OK|Exception)>
<!ELEMENT AddSessionListenerOnAllSessions (OK|Exception)>
<!ELEMENT SetProperty (OK|Exception)>
<!--ELEMENT Session (Property)*-->

<!--ATTLIST Session
      sid      CDATA #REQUIRED
      stype    (user|application) "user"
      cid      CDATA #REQUIRED
      cdomain  CDATA #REQUIRED
      maxtime  CDATA #REQUIRED
      maxidle  CDATA #REQUIRED
      maxcaching CDATA #REQUIRED
      timeleft CDATA #REQUIRED
      timeidle CDATA #REQUIRED
      state    (invalid|valid|inactive|destroyed) "invalid"-->
<!--ELEMENT Property-->
<!--ATTLIST Property
      name      CDATA #REQUIRED
      value     CDATA #REQUIRED-->
<!--ELEMENT SessionList (Session)*-->
<!--ELEMENT OK (#PCDATA)-->
<!--ELEMENT Exception (#PCDATA)-->
]>
```

Session Request XML

Code Example A-11 SessionRequest XML

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
<RequestSet vers="1.0" svcid="session" reqid="2">
  <Request>
    <![CDATA[<SessionRequest vers="1.0" reqid="1">
      <GetSession reset="true">
        <SessionID>fbaaa321f508529d@huanghe.eng.sun.com</SessionID>
      </GetSession>
    </SessionRequest>]]>
  </Request>
</RequestSet>
```

Session Response XML

Code Example A-12 SessionResponse XML

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
<ResponseSet vers="1.0" svcid="session" reqid="2">
  <Response>
    <![CDATA[<SessionResponse vers="1.0" reqid="1">
      <GetSession>
        <Session sid="fbaaa321f508529d@huanghe.eng.sun.com" stype="application"
          cid="authentication" cdomain="sun.com" maxtime="153722867280912930"
          maxidle="153722867280912930" maxcaching="153722867280912930"
          timeidle="0" timeleft="153722867280912930" state="valid">
        </Session></GetSession>
      </SessionResponse>]]>
    </Response>
  </ResponseSet>
```


Profile and Policy-related DTD and XML

Implementing the Profile and Policy API functionality in non-Java languages requires providing and parsing XML-based data streams to communicate with the Profile and Policy processes on the server.

NOTE If accessing Profile and Policy API functionality through Java programs, you need not be aware of these implementation specifics, because the Java interface makes these transparent to Java programmers. See the first section of this chapter to learn to access Profile and Policy API functionality through Java programs.

The following sections provide the DTDs that describe XML formats used for communicating with the Profile and Policy Service processes on the server. Non-Java programs must implement XML parsers to send requests to the server logging process in these formats as well as to interpret the XML-based responses from the server Profile and Policy process.

Code Example A-13 ProfileService DTD

```
<!ELEMENT ProfileService
(Message|GetProfile|SetProfile|DelProfile|
    GetChildRoles|SetParent|
    GetParent|GetUsers|Profile|Priv|Att|
    Exception|Data*|AddProfileListner|RemoveProfileListener|Notifica
tion)>
<!ATTLIST ProfileService
    reqid CDATA #REQUIRED
    ver CDATA #REQUIRED>

<!ELEMENT Message (#PCDATA)*>
<!ELEMENT Profile (Att*,Priv*)>
<!ATTLIST Profile
    profileName CDATA #REQUIRED
    profileType CDATA #REQUIRED>

<!ELEMENT Att
(Value*,Choice*,(ReadPermission?)*,(WritePermission?)*)>
<!ATTLIST Att
    attName CDATA #REQUIRED
    attId CDATA
    attMessage CDATA
    attType (string | number | boolean | singlechoice |
multichoice
```

Code Example A-13 ProfileService DTD (*Continued*)

```

        |protected | stringlist | numberlist ) "string"
remoteFlag (true|false) "false"
overrideFlag (true|false) "true"
inheritFlag (true|false) "false">
<!ELEMENT Value (#PCDATA)>
<!ELEMENT ReadPermission (#PCDATA)>
<!ELEMENT WritePermission (#PCDATA)>

<!ELEMENT Priv (Allow*,Deny*)>
<!--ATTLIST Priv
privName CDATA #REQUIRED
privId CDATA
privMessage CDATA
privType (boolean | list) #REQUIRED>
<!ELEMENT Allow (#PCDATA)>
<!ELEMENT Deny (#PCDATA)>

<!ELEMENT Exception (#PCDATA)>

<!ELEMENT Data (#PCDATA)>

<!--ELEMENT GetProfile (Profile)>
<!--ATTLIST GetProfile
admin (true|false) "false" #REQUIRED>

<!--ELEMENT SetProfile (Profile)>

<!--ELEMENT DelProfile (Profile)>

<!--ELEMENT GetChildRoles (Wildchar?)>
<!--ATTLIST GetChildRoles
parent CDATA #REQUIRED>

<!--ELEMENT GetParent EMPTY >
<!--ATTLIST GetParent
child CDATA #REQUIRED>

<!--ELEMENT SetParent (#PCDATA)>
<!--ATTLIST SetParent
child CDATA #REQUIRED>

<!--ELEMENT GetUsers (wildchar?)>
<!--ATTLIST GetUsers
parent CDATA #REQUIRED>

<!--ELEMENT AddProfileListner Empty>
<!--ATTLIST AddProfileListner
profileName CDATA #REQUIRED
listnerURL CDATA #REQUIRED>

<!--ELEMENT RemoveProfileListner Empty>
<!--ATTLIST RemoveProfileListner
profileName CDATA #REQUIRED
listnerURL CDATA #REQUIRED>

```

Code Example A-13 ProfileService DTD (*Continued*)

```
<!ELEMENT Wildchar (#PCDATA)>
<!ELEMENT Role (#PCDATA)>
```

Getting an Attribute Value Using XML

Code Example A-14 GetProfile Request XML

```
Profile request: get iwtPlatform-locale value

<ProfileService ver="1.0" reqid="0"><GetProfile
searchFlag="false" admin="false"><Profile
profileName+"/component/iwtPlatform" profileType="7"><Att
attName="iwtPlatform-locale"></Att></Profile><?GetProfile></Prof
ileService>
```

Code Example A-15 GetProfile Response XML

```
Profile request: get iwtPlatform-locale value

<ProfileService ver="1.0" reqid="0"><Profile><Att
attName="iwtPlatform-locale" attType="string" attId=""
attDes="Platform Locale" inheritFlag="true" overrideFlag="false"
remoteFlag="false"
emptyFlag="false"><Value>en_US</Value><ReadPermission>ADMIN</Rea
dPermission>OWNER</ReadPermission><WritePermission>ADMIN</WriteP
ermission></Att></Profile></ProfileService>
```

Log-related DTDs

Implementing the Log API functionality in non-Java languages requires providing and parsing XML-based data streams to communicate with the logging processes on the server.

NOTE If accessing Log API functionality through Java programs, you need not be aware of these implementation specifics, because the Java interface makes these transparent to Java programmers. See the first section of this chapter to learn to access Log API functionality through Java programs.

The following sections detail the DTDs that describe acceptable communication with the logging processes.

DTD for Log API Communication

Code Example A-16 DTD for Log API Communication

```
<?XML version="1.0">
<!--This DTD is used by Logging operation-->
<!DOCTYPE logging[

<!ELEMENT logCreate (log)>
<!ELEMENT logDelete (log)>
<!ELEMENT logRecWrite (log, logRecord*)>
<!ELEMENT logRecRead (log, queryString?)>
<!ELEMENT logList (log*)>

<!ELEMENT log>
<!ATTLIST log
    logName          CDATA #REQUIRED
    status            CDATA #IMPLIED
    maxFileSize       CDATA #IMPLIED
    numHistoryFile    CDATA #IMPLIED
    location          CDATA #IMPLIED
>

<!ELEMENT queryString (#PCDATA)>

<!ELEMENT logRecord (recType, recMsg)>
<!ELEMENT recType (#PCDATA)>
<!ELEMENT recMsg (#PCDATA)>

]
```

Putting Code Together

This Appendix describes the development process for a sample iPlanet Portal Server desktop provider application that touches on the public APIs available for integrating an application with the iPlanet Portal Server desktop.

Building an iPlanet Portal Server Provider

Start by taking the `HelloWorldProvider.java` and reducing it to the bare minimum. For example, change the class name to `QuotationProvider` and leave the shell of the essential routines:

Table B-1 Minimal Routines for a Provider

Routine	Purpose
<code>QuotationProvider()</code>	The constructor
<code>getHTMLContent()</code>	Used to display the provider's HTML on the desktop
<code>getHTMLEditForm()</code>	Used to display an HTML form where the user can set preferences.
<code>processEditForm()</code>	Used to process the submitted form created in <code>getHTMLEditForm</code> above.

Define Specific Requirements and Functionality

For this example, the application should handle *Quotation display*, as described in the following section:

Depending on a user-specific configuration parameter either:

- One quote of each user-specified category will be displayed
- Only one quote from all of the user-specified categories will be displayed

Depending on a user-specific configuration parameter, one of the following will be displayed:

- The category of, and the quote
- Only the quote itself

For the “Edit” part of the provider, where the user can edit configuration parameters, the following will displayed:

- A multiple-choice selection of the types of quotes to be displayed
- A checkbox to choose whether categories get displayed as well or not

In addition to the user-selectable configuration parameters, only an administrator should be able to change the location of the quotations file (via the Administration Console).

Identify non-iPlanet Portal Server Functionality

In this example, all of the quotations are in a file of the following format:

```
quote-type|quotation-text
```

For example:

```
Computers|blah blah blah - Scott McNealy  
Politics|blah blah blah - Genghis kahn
```

where the category of the quotation is separated from the quotation itself by a vertical bar '|'. For the sake of this example, the file must be located in `/var/tmp/quotations`. This can be overridden by an iPlanet Portal Server administrator but not by the individual user.

Define Application Attributes/Privileges

For an example, adopt a naming scheme that parallels the default providers that come with iPlanet Portal Server, and name the provider *iwtQuotationProvider*. The attributes will be called

:

Table B-2 Sample Provider Attributes

Attribute	Description
iwtQuotationProvider-possibleCategories	A multi valued set of strings representing the possible categories of quotes a specific user would be interested in. In the following example, the quotation categories will be limited to “Computers”, “Science”, and “Freedom”.
iwtQuotationProvider-selectedCategories	A multi valued set of strings representing the categories of quotes a specific user would be interested in, such as “Computers”, “Science”, and “Freedom”.
iwtQuotationProvider-displayCategories	A boolean value indicating whether categories should be displayed with the quotations or not.
iwtQuotationProvider-fileLocation	A single string value indicating where the list of quotations is to be found.

Define the Provider to iPlanet Portal Server

The provider must be registered with the iPlanet Portal Server before the Session, Profile, or Logging APIs may be used. To do this, write (or adapt) an XML file to describe the attributes and privileges, then copy the XML file to `/etc/opt/SUNWips/xml`, and use `ipsadmin` to import the XML file into the Profile server. See Chapter 3, “Profile and Policy API” for more details of this function.

The easiest way to begin is to adapt the example `helloWorld3` provider's XML file. Take it and modify the names of the component and attributes to be `iwtQuotationProvider`. The only attributes/privileges in the XML file at this point are the “common” attributes which any provider must have.

Next, add in the attributes specific to this application:

Code Example B-1 XML Defining Attributes and Privileges

```

<!--
  Attributes, specific
-->

<iwt:Att name="iwtQuotationProvider-possibleCategories"
  desc="Possible Quotation Categories"
  type="stringlist"
  idx="possible_quotation_categories"
  userConfigurable="TRUE">
  <Val>Computers</Val>
  <Val>Freedom</Val>
  <Val>Science</Val>
  <Val>Work</Val>
  <Rperm>ADMIN</Rperm><Rperm>OWNER</Rperm>
  <Wperm>ADMIN</Wperm>
</iwt:Att>

<iwt:Att name="iwtQuotationProvider-selectedCategories"
  desc="Selected Quotation Categories"
  type="stringlist"
  idx="selected_categories"
  userConfigurable="TRUE">
  <Val>Computers</Val>
  <Val>Freedom</Val>
  <Val>Science</Val>
  <Val>Work</Val>
  <Rperm>ADMIN</Rperm><Rperm>OWNER</Rperm>
  <Wperm>ADMIN</Wperm><Wperm>OWNER</Wperm>
</iwt:Att>

<iwt:Att name="iwtQuotationProvider-displayCategories"
  desc="Display category along with quotation?"
  type="boolean"
  idx="display_categories"
  userConfigurable="TRUE">
  <Val>true</Val>
  <Rperm>ADMIN</Rperm><Rperm>OWNER</Rperm>
  <Wperm>ADMIN</Wperm><Wperm>OWNER</Wperm>
</iwt:Att>

<iwt:Att name="iwtQuotationProvider-fileLocation"
  desc="Location of the quotations file"
  type="string"
  idx="file_location"
  userConfigurable="TRUE">
  <Val>/var/tmp/quotations</Val>
  <Rperm>ADMIN</Rperm><Rperm>OWNER</Rperm>
  <Wperm>ADMIN</Wperm>
</iwt:Att>

```


Attribute by attribute, Table B-3 describes what the XML in the code example means:

Table B-3 Sample Attributes and Privileges

Attribute	Description
<code>iwtQuotationProvider-possibleCategories</code>	This is a stringlist attribute that the ADMIN and OWNER can read, but only the ADMIN can write. It is preset to use the 3 quotation categories mentioned above, in this example.
<code>iwtQuotationProvider-selectedCategories</code>	This is a stringlist attribute that the ADMIN and OWNER can read and write. It is preset to the same categories as the previous attribute.
<code>iwtQuotationProvider-displayCategories</code>	This is a boolean attribute that the ADMIN and OWNER can read and write. It is preset to “true” indicating that category names will be displayed with the quotes.
<code>iwtQuotationProvider-fileLocation</code>	This is a single string attribute that the ADMIN and OWNER can read, but only the ADMIN can write. It is preset to <code>/var/tmp/quotations</code> .

Sample Code

The following code shows how the entire application came together.

Code Example B-2 Sample Desktop Provider File

```
// QuotationProvider.java
// This is a sample desktop provider application used to display
// famous/humorous quotations on the desktop.
//
// It shows how to use the ProfileProviderAdapter API to:
// - display HTML content on the desktop
// - allow the user to edit attributes
//
// It also invokes the Session, Profile and Logging APIs in
// order to:
// - get/set user-level attributes regarding quotation
// preferences
// - provide logging

// Where the QuotationProvider.class file can be found.
package com.iplanet.portalserver.providers.quotations;
```

Code Example B-2 Sample Desktop Provider File (*Continued*)

```
// QuotationProvider.java

// Use some basic Java classes
import java.lang.*;
import java.io.*;
import java.util.*;

// Use some basic Java Servlet classes
import javax.servlet.*;
import javax.servlet.http.*;

// Use some iPS classes
import com.iplanet.portalserver.desktop.util.*;
import com.iplanet.portalserver.providers.Provider;
import com.iplanet.portalserver.providers.ProfileProviderAdapter;
import com.iplanet.portalserver.session.Session;
import com.iplanet.portalserver.profile.*;

// The QuotationProvider Class
public class QuotationProvider extends ProfileProviderAdapter
implements Provider {

    // The constructor - don't need to do any special
    initialization
    // so just invoke the ProfileProviderAdapter's constructor.
    public QuotationProvider() {
        super();
    }

    // Generate the HTML that will be displayed in this content
    provider's
    // area on the desktop
    public StringBuffer getHTMLContent() throws Exception {
        Profile p = getSession().getUserProfile();
        StringBuffer content = new StringBuffer(); // Contains
        HTML to be displayed.
        Hashtable quotesHash = new Hashtable(); // A Hash of
        quote Vectors, one vector per category.
        BufferedReader quotesReader; // Used to read
        the quotes.
        String quote; // A single
        quotation.
        Random randomGenerator = new Random(); // To randomly
        pick quotes.

        // Read in the quotations into a Hashtable of Vectors,
        each containing
        // the quotations of a specific category. If we can't read
        the
        // quotations file it's okay - we just won't display any
        quotes later.
    }
}
```

Code Example B-2 Sample Desktop Provider File (*Continued*)

```

// QuotationProvider.java
// The category and quotation are separated by a "|" in
the quotations
// file.
try {
    String quotationFileName =
p.getAttributeString("iwtQuotationProvider-fileLocation");
    quotesReader = new BufferedReader(new FileReader(new
File(quotationFileName)));
    while ((quote = quotesReader.readLine()) != null) {
        String type =
quote.substring(0,quote.indexOf('|'));
        if (!quotesHash.containsKey(type)) {
            quotesHash.put(type,new Vector());
        }

        ((Vector)(quotesHash.get(type))).addElement(quote.substring(quot
e.indexOf('|')+1));
    }
} catch (Exception e) {

    // Determine if user's preference is to display the
category along with a quotation.
    boolean displayCategories = true;
    if
(p.getAttributeString("iwtQuotationProvider-displayCategories").
equals("false")) {
        displayCategories = false;
    }

    // Get the possible quotation categories.
    Enumeration e =
p.getAttribute("iwtQuotationProvider-selectedCategories");

    // Print a quotation for each category selected by the
user.
    while (e.hasMoreElements()) {
        String category = (String)e.nextElement();

        // If the user wanted to display the category along
with the
        // quotation then do so.
        if (displayCategories) {
            content.append("<BOLD>");
            content.append(category);
            content.append("</BOLD>");
        }

        // If there are any quotations for this category, pick
a random
        // one and display it, otherwise note that we didn't
find one.

```

Code Example B-2 Sample Desktop Provider File (*Continued*)

```

// QuotationProvider.java
    if (quotesHash.containsKey(category)) {
        Vector quoteVector =
        (Vector)quotesHash.get(category);
        int whichQuoteIndex =
        Math.abs(randomGenerator.nextInt())%(quoteVector).size();
        content.append("<UL>");
        content.append("<LI>");

        content.append((String)(quoteVector.elementAt(whichQuoteIndex)))
        ;

        content.append("</LI>");
        content.append("</UL>");
    } else {
        content.append("<UL>");
        content.append("<LI>");
        content.append("(no quotes of this type found)");
        content.append("</LI>");
        content.append("</UL>");
    }
}

// Now return the HTML we have constructed.
return content;
}

// Generate the HTML that will be displayed to let the user
set his preferences.
public StringBuffer getHTMLEditForm() {
    StringBuffer content = new StringBuffer();

    try {
        Profile p = getSession().getUserProfile();

        // Display things in a single column table - so things
line up.
        content.append("<P><TABLE>");

        // Let the user choose which categories he wants.
        content.append("<TR>");
        content.append("<TD VALIGN='top' HALIGN='left'>");
        content.append("Types of Quotes to Display");
        content.append("</TD>");

        // Get the user's selected categories
        Vector selectedCategories = new Vector();
        Enumeration e2 =
        p.getAttribute("iwtQuotationProvider-selectedCategories");
        while (e2.hasMoreElements()) {

            selectedCategories.addElement((String)e2.nextElement());
        }
    }
}

```

Code Example B-2 Sample Desktop Provider File (*Continued*)

```

// QuotationProvider.java
// List the possible categories, marking those that
have already
// been selected by the user.
content.append("<TD VALIGN='top' HALIGN='left'>");
Enumeration e =
p.getAttribute("iwtQuotationProvider-possibleCategories");
while (e.hasMoreElements()) {
    String category = (String)e.nextElement();
    content.append("<INPUT TYPE='checkbox'
NAME='category-");
    content.append(category);
    content.append("' VALUE='");
    content.append(category);
    content.append("'");
    if (selectedCategories.contains(category)) {
        content.append(" CHECKED");

        content.append(">");
        content.append(category);
        content.append("<BR>");
    }
    content.append("</TD>");
    content.append("</TR>");

    // Let the user decide if he wants to display a
category with each quotation.
    content.append("<TR>");
    content.append("<TD VALIGN='top' HALIGN='left'>");
    content.append("<INPUT TYPE='checkbox'
NAME='display_categories' VALUE='yes'");
    if
(p.getAttributeString("iwtQuotationProvider-displayCategories").
equals("true")) {
        content.append(" CHECKED");
    }
    content.append(">Display the Category Along With the
Quotation");
    content.append("</TD>");
    content.append("</TR>");

    content.append("</TABLE>");
} catch (Exception e) {

    return content;
}

// Handle the submittal of the edit form.
public int processEditForm(HttpServletRequest req) throws
Exception {
    Profile p = getSession().getUserProfile();
    Enumeration parameterNames = req.getParameterNames();
    Vector categorySelections = new Vector();

```

Code Example B-2 Sample Desktop Provider File (*Continued*)

```

// QuotationProvider.java
boolean displayCategories = false;

        // Check all the passed parameters and add the selected
categories
        // as well as check to see if user opted to display
categories.
        while (parameterNames.hasMoreElements()) {
            String parameter =
(String)parameterNames.nextElement();
            if (parameter.equals("display_categories")) {
                displayCategories = true;
            } else if (parameter.startsWith("category-")) {
                String category = req.getParameter(parameter);
categorySelections.addElement(category);
            }
        }

        // Set the list of selected categories for later storing
in the profile database.
        p.setAttribute("iwtQuotationProvider-selectedCategories",
categorySelections.elements(), Profile.NEW);

        // Set whether the user wants to display categories or not
for later
        // storing in the profile database.
        if (displayCategories) {

p.setAttributeString("iwtQuotationProvider-displayCategories",
"true", Profile.NEW);
        } else {

p.setAttributeString("iwtQuotationProvider-displayCategories",
"false", Profile.NEW);
        }

        // Done processing the edit form - now store the user's
preferences.
        p.store(true);

        return DtConstants.BUILD_FRONT;
    }
}

```

The sample below is the complete XML file used to register this sample provider with the Profile and Policy Server.

Code Example B-3 Sample XML File

```

<!--
Copyright 11/30/99 Sun Microsystems, Inc. All Rights Reserved.
"@(#)iwtQuotationProvider.xml 1.2 99/11/30 Sun Microsystems"
-->

<iwt:Component name="iwtQuotationProvider"
  ver="1.0"
  desc="Quotation Provider"
  resB="iwtQuotationProvider.rb"
  idx="iwtQuotationProvider-desc" >

  <!--
  Attributes, common
  -->

  <iwt:Att name="iwtQuotationProvider-title"
    desc="Title for this provider"
    type="string"
    idx="title-desc"
    userConfigurable="TRUE">
    <Val>Quotation Provider</Val>
    <Rperm>ADMIN</Rperm><Rperm>OWNER</Rperm>
    <Wperm>ADMIN</Wperm>
  </iwt:Att>
  <iwt:Att name="iwtQuotationProvider-width"
    desc="Width for this provider"
    type="singlechoice"
    idx="width-desc"
    userConfigurable="TRUE">
    <Val>thick</Val>
    <CVal>thin</CVal><CVal>thick</CVal>
    <Rperm>ADMIN</Rperm><Rperm>OWNER</Rperm>
    <Wperm>ADMIN</Wperm>
  </iwt:Att>
  <iwt:Att name="iwtQuotationProvider-isDetached"
    desc="Is this provider detached from the desktop?"
    type="boolean"
    idx="isDetached-desc"
    userConfigurable="TRUE">
    <Val>false</Val>
    <Rperm>ADMIN</Rperm><Rperm>OWNER</Rperm>
    <Wperm>ADMIN</Wperm><Wperm>OWNER</Wperm>
  </iwt:Att>
  <iwt:Att name="iwtQuotationProvider-isMinimized"
    desc="Is this provider minimized on the desktop?"
    type="boolean"
    idx="isMinimized-desc"
    userConfigurable="TRUE">
    <Val>false</Val>
    <Rperm>ADMIN</Rperm><Rperm>OWNER</Rperm>
    <Wperm>ADMIN</Wperm><Wperm>OWNER</Wperm>
  </iwt:Att>
  <iwt:Att name="iwtQuotationProvider-debug"

```

Code Example B-3 Sample XML File (*Continued*)

```

        desc="Is debugging output on for this provider?"
        type="boolean"
        idx="debug-desc"
        userConfigurable="FALSE">
        <Val>>false</Val>
        <Rperm>ADMIN</Rperm><Rperm>OWNER</Rperm>
        <Wperm>ADMIN</Wperm>
    </iwt:Att>
    <iwt:Att name="iwtQuotationProvider-helpLink"
        desc="Help page for this provider"
        type="string"
        idx="helpLink-desc"
        userConfigurable="TRUE">
        <Val>user_help/desktop/desktopTOC.html</Val>
        <Rperm>ADMIN</Rperm><Rperm>OWNER</Rperm>
        <Wperm>ADMIN</Wperm>
    </iwt:Att>
    <iwt:Att name="iwtQuotationProvider-templateKeys"
        desc="Template keys for this provider"
        type="stringlist"
        idx="templateKeys-desc"
        userConfigurable="TRUE">
        <Rperm>ADMIN</Rperm><Rperm>OWNER</Rperm>
        <Wperm>ADMIN</Wperm>
    </iwt:Att>
    <iwt:Att name="iwtQuotationProvider-templateFiles"
        desc="Template files for this provider"
        type="stringlist"
        idx="templateFiles-desc"
        userConfigurable="TRUE">
        <Rperm>ADMIN</Rperm><Rperm>OWNER</Rperm>
        <Wperm>ADMIN</Wperm>
    </iwt:Att>
    <iwt:Att name="iwtQuotationProvider-column"
        desc="Column to display this provider in"
        type="string"
        idx="column-desc"
        userConfigurable="TRUE">
        <Val>1</Val>
        <Rperm>ADMIN</Rperm><Rperm>OWNER</Rperm>
        <Wperm>ADMIN</Wperm><Wperm>OWNER</Wperm>
    </iwt:Att>

    <!--
    Attributes, specific
    -->

    <iwt:Att name="iwtQuotationProvider-possibleCategories"
        desc="Possible Quotation Categories"
        type="stringlist"
        idx="possible_quotation_categories"
        userConfigurable="FALSE">
        <Val>Computers</Val>
        <Val>Freedom</Val>
        <Val>Science</Val>

```


Code Example B-3 Sample XML File (*Continued*)

```

        <Val>Stupid</Val>
        <Val>Work</Val>
        <Rperm>ADMIN</Rperm><Rperm>OWNER</Rperm>
        <Wperm>ADMIN</Wperm><Wperm>OWNER</Wperm>
    </iwt:Att>

    <iwt:Att name="iwtQuotationProvider-selectedCategories"
        desc="Selected Quotation Categories"
        type="stringlist"
        idx="selected_categories"
        userConfigurable="TRUE">
        <Val>Computers</Val>
        <Val>Freedom</Val>
        <Val>Science</Val>
        <Val>Stupid</Val>
        <Val>Work</Val>
        <Rperm>ADMIN</Rperm><Rperm>OWNER</Rperm>
        <Wperm>ADMIN</Wperm><Wperm>OWNER</Wperm>
    </iwt:Att>

    <iwt:Att name="iwtQuotationProvider-displayCategories"
        desc="Display category along with quotation?"
        type="boolean"
        idx="display_categories"
        userConfigurable="TRUE">
        <Val>true</Val>
        <Rperm>ADMIN</Rperm><Rperm>OWNER</Rperm>
        <Wperm>ADMIN</Wperm><Wperm>OWNER</Wperm>
    </iwt:Att>

    <iwt:Att name="iwtQuotationProvider-fileLocation"
        desc="Location of the quotations file"
        type="string"
        idx="file_location"
        userConfigurable="FALSE">
        <Val>/var/tmp/quotations</Val>
        <Rperm>ADMIN</Rperm><Rperm>OWNER</Rperm>
        <Wperm>ADMIN</Wperm>
    </iwt:Att>

    <!--
    Privileges, common
    -->

    <iwt:Priv name="iwtQuotationProvider-isMinimizable"
        desc="Can this provider be minimized on the desktop?"
        type="boolean"
        idx="isMinimizable-desc"
        userConfigurable="TRUE"
        val="true">
        <Rperm>ADMIN</Rperm><Rperm>OWNER</Rperm>
        <Wperm>ADMIN</Wperm>
    </iwt:Priv>
    <iwt:Priv name="iwtQuotationProvider-isDetachable"
        desc="Can this provider be detached from the desktop?"

```

Code Example B-3 Sample XML File (*Continued*)

```

        type="boolean"
        idx="isDetachable-desc"
        userConfigurable="TRUE"
        val="true">
        <Rperm>ADMIN</Rperm><Rperm>OWNER</Rperm>
        <Wperm>ADMIN</Wperm>
    </iwt:Priv>
    <iwt:Priv name="iwtQuotationProvider-hasHelp"
        desc="Does this provider have a help page?"
        type="boolean"
        idx="hasHelp-desc"
        userConfigurable="TRUE"
        val="true">
        <Rperm>ADMIN</Rperm><Rperm>OWNER</Rperm>
        <Wperm>ADMIN</Wperm>
    </iwt:Priv>
    <iwt:Priv name="iwtQuotationProvider-isEditable"
        desc="Does this provider have an edit page?"
        type="boolean"
        idx="isEditable-desc"
        userConfigurable="TRUE"
        val="true">
        <Rperm>ADMIN</Rperm><Rperm>OWNER</Rperm>
        <Wperm>ADMIN</Wperm>
    </iwt:Priv>
    <iwt:Priv name="iwtQuotationProvider-isRemovable"
        desc="Can this provider be removed from the desktop?"
        type="boolean"
        idx="isRemovable-desc"
        userConfigurable="TRUE"
        val="true">
        <Rperm>ADMIN</Rperm><Rperm>OWNER</Rperm>
        <Wperm>ADMIN</Wperm>
    </iwt:Priv>
    <iwt:Priv name="iwtQuotationProvider-hasBorder"
        desc="Should this provider be drawn with a border around
it?"
        type="boolean"
        idx="hasBorder-desc"
        userConfigurable="TRUE"
        val="true">
        <Rperm>ADMIN</Rperm><Rperm>OWNER</Rperm>
        <Wperm>ADMIN</Wperm>
    </iwt:Priv>
</iwt:Component>

```

The following code sample shows the sample quotations file used for this example.

Code Example B-4 Sample Quotation File

```

Computers|"If you can't beat your computer at chess, try
kickboxing." -Anon.
Computers|"What goes up must come down. Ask any system
administrator." -Anon.
Computers|"Perl - The only language that looks the same before
and after RSA encryption." -Keith Bostic
Computers|"C makes it easy to shoot yourself in the foot. C++
makes it harder, but when you do, it blows away your whole leg."
-Bjarne Stroustrup
Freedom|"Live free or die." -New Hampshire State Motto
Freedom|"Man is free at the moment he wishes to be." -Voltaire
Freedom|"People demand freedom of speech as a compensation for
the freedom of thought which they seldom use." -Kierkegaard
Freedom|"The right to be heard does not automatically include the
right to be taken seriously." -Hubert Humphrey
Freedom|"Extremism in the defense of liberty is no vice. And
moderation in the pursuit of justice is no virtue." -Barry
Goldwater
Freedom|"I disapprove of what you say, but I will defend to the
death your right to say it." -Voltaire
Science|"Give me a lever long enough and a fulcrum on which to
place it, and I shall move the world." -Archimedes, Pappus of
Alexandria
Science|"A scientific truth does not triumph by convincing its
opponents and making them see the light, but rather because its
opponents eventually die and a new generation grows up that is
familiar with it." -Maxwell Planck
Science|"Every great advance in science has issued from a new
audacity of imagination." -John Dewey, The Quest for Certainty

```


iPlanet Portal Server API Exceptions

Profile API Exceptions

Table C-1 Profile API Exceptions

Exception Type	Exception Description
100	Multivalued Attribute
101	No Attributes in profile
102	User role attribute not set
103	Not Boolean type privilege
104	Not List type privilege
105	Not valid match
106	No Privileges in profile
107	Privilege not found
108	Attribute not found
118	Invalid session
119	Not found
122	Invalid wildchar
300	Fatal exception initializing Data Store
301	Required data store initialization parameters not present
310	Invalid attribute value
311	Invalid privilege value

Table C-1 Profile API Exceptions (*Continued*)

Exception Type	Exception Description
312	Invalid choice value
320	Invalid profile name
321	Invalid attribute or privilege name
322	Invalid parent profile name
323	Profile name already in use
324	Unable to create profile
325	Unable to remove profile
326	Invalid profile type
330	Unable to get attribute or privilege value from data store
331	Profile is not initialized
332	Unable to remove attribute from data store
333	Invalid operation for modifying attribute value
334	Unable to modify attribute value in data store
340	Invalid search filter
341	Search failed
700	No parent profile defined
701	No role defined for user
702	Profile does not exist
703	Undefined attribute or privilege
705	Invalid data
710	Can't assign multi-value to single-value type attribute
711	Attribute is not configurable
720	Permission denied in getting attributes or privileges
721	Permission denied in setting attributes or privileges
722	Permission denied in creating profile
723	Permission denied in deleting profile
724	Permission denied
725	Set profile failed in overwriting customized attributes in sub profiles

Table C-1 Profile API Exceptions (*Continued*)

Exception Type	Exception Description
726	Permission denied in searching profile
740	Invalid attribute name or value provided in search profile
741	Invalid profile type provided in search profile
760	Invalid external attribute mapping
761	Unable to get external LDAP configuration attributes
762	Invalid search scope
763	Unable to read from external data store
764	Unable to write to external data store
765	Unable to connect to external data store
766	Unable to close external data store connection
767	Duplicate matches in external data store
770	Invalid external LDAP connection properties
800	Invalid Session Exception
850	Invalid Notification URL
851	Unable to remove profile service listener

Log API Exceptions

Table C-2 Logging API Exceptions

Exception Type	Exception Description
500	INVALID_SESSION
501	ALREADY_EXISTS
502	INACTIVE
503	LOG_HANDLER_ERROR
504	WRITE_ERROR
505	READ_ERROR

Table C-2 Logging API Exceptions *(Continued)*

Exception Type	Exception Description
506	DELETE_ERROR
507	LIST_NOT_EXISTS
508	TYPE_ERROR
509	CREATE_ACCESS_DENIED
510	WRITE_ACCESS_DENIED
511	READ_ACCESS_DENIED
512	LIST_ACCESS_DENIED
513	PROFILE_ERROR
514	LOG_NOT_FOUND
515	NO_SUCH_SEGMENT_EXISTS
516	DELETE_ACCESS_DENIED
517	INVALID_LOG_NAME
518	READ_EXCEEDS_MAX
519	DRIVER_LOAD_FAILED
520	NULL_LOCATION
521	CONNECTION_FALIED
522	NULL_POINTER
523	SQL_ERROR
699	FATAL_ERROR

Session API Exceptions

Table C-3 Session API Exceptions

Exception Type	Exception Descreption
unexpectedResponse	Unexpected number of responses received
unexpectedSession	Unexpected number of sessions received

Table C-3 Session API Exceptions

Exception Type	Exception Description
invalidSessionID	Invalid session ID
invalidSessionState	Invalid session state
noPrivilege	No privilege to perform this operation

Glossary

access control Implements the privileges granted by authorization.

address In networking, a unique code that identifies a *node* to the *network*. Names like i-planet.demo.sun.com are translated to “dotted quad” addresses (10.0.24.15) by the Domain Name Service. (DNS).

administration console The administrator’s GUI interface to iPlanet Portal Server 3.0.

API Application Program Interface, a set of calling conventions or instructions defining how programs invoke services in existing software packages.

applet A program written in the Java™ programming language to run within a Web browser. An example would be the Java front ends to i-Planet’s NetMail and NetFile applications.

attribute A configurable parameter of a profile.

ASP Access Service Provider. A company that, for a fee, provides access to applications that users can run without owning their own copies. See *ISP*.

authentication The process of verifying a user’s identity.

authentication module An authentication module controls a specific authentication process. For example, i-Planet provides authentication modules for Microsoft Windows NT, UNIX, S/key, and others, as well as opening the authentication API so other authentication modules can be written as needed.

authorization The process of granting specific access privileges to a user. Authorization is based on authentication and enforced by access control.

CA See *Certificate Authority*.

cache In Web browsers, the archive of recently visited Web pages, graphics, or other files that is stored in memory or on users' disks.

CDP Certificate Discovery Protocol. Request and response protocol used by two parties to transfer certificates.

certificate A set of data that identifies a person, machine, or application.

certificate identifier (ID) Generic naming scheme term used to identify a particular self-generated or issued certificate. It effectively decouples the identification of a key for purposes of key lookup and access control from issues of network topology, routing, and IP addresses.

Certificate Authority (CA) Trusted network entity that digitally signs a certificate containing information identifying the user; such as the user's name, the issued certificate, and the certificate's expiration date. Verisign is one of the best known CA's.

component An application or a service in i-Planet. Components have attributes and privileges, much like users.

content filtering Practice of allowing or disallowing traffic based on the content of the data being sent.

cookie General mechanism that server-side connections can use to store and retrieve information on the client side of the connection. Cookies are small data files written to a user's hard drive by some Web sites when viewed in a Web browser. These data files contain information the site can use to track such things as passwords, lists of pages visited, and the date when a certain page was last looked at.

data compression Application of an algorithm to reduce the space required to store or the bandwidth required to transmit data.

decryption Process of decrypting information that has been encrypted. See *encryption*.

demilitarized zone (DMZ) Small protected network between the public Internet and a private intranet, usually demarcated with firewalls on both ends. This area is used to provide limited public access to resources such as Web servers, FTP servers, and other information resources.

desktop What the user sees on the screen. This usually includes a preferred set of applications and access privileges.

digital signatures Data added to a document to identify the sender using a public-key encryption scheme.

DMZ See *demilitarized zone*.

DNS Domain Name Service is a distributed name and address lookup mechanism used to translate domain names (ips.demo.sun.com) to IP addresses (10.23.134.24). It also allows reverse lookup, to translate IP addresses back into names.

domain The last part of a *fully qualified domain name* that identifies the company or organization that owns the domain name (for example, sun.com, sun.co.uk).

encryption Process of protecting information from unauthorized use by making the information unintelligible. Some encryption methods employ codes, called keys, which are used to encrypt the information. Contrast with *decryption*.

firewall Computer situated between an internal network and the rest of the network that filters packets as they go by according to user-specified criteria. Firewalls are normally used to protect systems on one side from unauthorized access by users on the other side.

File Transfer Protocol (FTP) A file transfer protocol often used on TCP/IP networks to copy files to and from remote computers.

fully qualified domain name The complete domain name of a system, including the hostname, network name if applicable, and domain; for example west.sun.com.

gateway A system that provides and controls connections to another network. See *VPN*.

host Name of a device on a TCP/IP network that has an IP address.

HTML Hypertext Markup Language. A file format, based on SGML, for hypertext documents on the Internet.

HTTP Hypertext Transfer Protocol, which describes how Web browsers and Web servers exchange information. See *URL*.

HTTPS Hypertext Transfer Protocol Secure, which describes the use of HTTP over an SSL connection, usually on port 443.

ICMP Internet Control Message Protocol. IP protocol that handles errors and control messages, to enable routers to inform other routers (or hosts) of IP routing problems or make suggestions of better routes. See *ping*.

IMAP Internet Message Access Protocol allows remote access to mailboxes and folders. IMAP clients usually leave some or all messages and folders on the server, unlike POP, in which all messages are downloaded.

Internet Protocol Protocol within TCP/IP suite used to link networks worldwide, developed by the United States Department of Defense and is used on the Internet. The prominent feature of this suite is the IP protocol.

IP See *Internet Protocol*.

ISP Internet Service Provider. A company providing Internet access. This service often includes a phone number access code, username, and software—all for a provider fee.

issued certificate Certificate that is *issued* by a *Certificate Authority*. See *self-generated certificate*.

ISV Independent Software Vendor. Third-party software developer.

Java™ Object-oriented, platform independent programming language developed by Sun Microsystems to solve a number of problems in modern programming practice.

JDK Java Development Kit. Software tools used to write Java applets or application programs.

key Code for encrypting or decrypting data.

LAN Local area network, a private network at a single location. Multiple LANs can be interconnected to form a WAN.

LDAP Lightweight Directory Access Protocol. One of the protocols used in iPlanet Portal Server 3.0 to resolve profile attributes and privileges.

load balancer A load balancer controls connections to multiple gateway machines to allow approximately equivalent loads on each of the available systems.

NAT See *network address translation*.

Netlet A Java applet used in i-Planet to allow any TCP/IP-based applications to securely connect to servers through an authenticated iPS connection.

network address translation (NAT) Function used when packets passing through a firewall have their addresses changed (or translated) to different network addresses. Address translation can be used to translate unregistered addresses into a smaller set of registered addresses, thus allowing internal systems with unregistered addresses to access systems on the Internet.

network mask Number used by software to separate the local subnet address from the rest of a given IP address.

NFS™ Network File System. A file system distributed by Sun Microsystems that enables a set of computers to cooperatively access each others files in a transparent manner.

NIS and NIS+ Network Information Service. NIS+ is a newer version (with a lookup service) for Solaris 2.x, with enhanced security.

node A transfer point within a network. Data is passed from node to node in a network until the data reaches its final destination.

passphrase Collection of characters used in a similar manner to, although typically longer than, a password. See *password*.

password Unique string of characters that a user types as an identification code; a security measure to restrict access to computer systems and sensitive files.

personal digital certificate (PDC) An electronic certificate attached to a message that authenticates a user. A personal digital certificate can be created by correctly entering a userID and password, or by using an SSL certificate request that in turn uses the security certificate of the server through which the user is connected.

PDC See *personal digital certificate*.

ping A TCP/IP command that verifies a connection to another host.

plaintext Unencrypted message.

Point-to-Point Protocol (PPP) PPP (the successor to SLIP) provides router-to-router and host-to-network connections over both synchronous and asynchronous circuits. Used for TCP/IP connectivity, usually for PC's over a telephone line. Also known as PPTP.

POP Post Office Protocol; defines a mechanism with which Internet users can connect to and download their waiting email messages.

PPP See *Point-to-Point Protocol*.

port The location (or socket) to which TCP/IP connections are made. Web servers traditionally use port 80, while FTP uses port 21 and telnet uses port 23. i-Planet uses some special ports, particularly on client systems, to securely communicate through the iPS session to servers.

preference A user-specified choice about what appears or doesn't appear on the desktop, and how it appears, or other traits such as timeout settings.

private network A network of computers that is inaccessible unless you have appropriate access privileges. Private networks may be as small as a one-office LAN or as large as a multi-country enterprise network. See also *public network*.

privilege A type of access right that is granted to a user, a set of users, or a resource that is specified by the particular type of authorization implemented.

profile The attributes and privileges for an iPS entity, such as user, role, domain, or component.

profile server A special segment of iPlanet Portal Server 3.0 that is devoted to storing profile information.

protocol A formal description of messages to be exchanged and rules to be followed for two or more systems to exchange information.

proxy A proxy is an intermediary program that makes and services requests on behalf of clients. Proxies act as servers and clients in turn, and are used to control the content of various network services. See *reverse proxy*.

public-key certificate A data structure containing a user's public key, as well as information about the time and date during which the certificate is valid.

public-key cryptography Also known as *asymmetric* key cryptography. In public-key cryptosystems, everyone has two related complementary keys: a publicly revealed key and a *secret* key (also frequently called a *private* key). Each key unlocks the code that the other key makes. Knowing the public key does not help you deduce the corresponding secret key. The public key can be published and widely disseminated across a communications network. This protocol provides privacy without the need for the secure channels that a conventional cryptosystem requires.

public network Like the Internet, a public network carries traffic from a variety of companies, individuals, and sources and is inherently insecure. Contrast with *private network*.

query Process for extracting particular data.

reverse proxy A proxy which performs bi-directional URL rewriting and translation between clients and servers. Unlike a proxy, which exists at the client side, a reverse proxy exists at the server side of the network. In i-Planet, the reverse proxy exists on the iPS gateway.

role A role defines all aspects of a user's experience when running in the i-Planet environment. A role can, for instance, correspond to a job title (manager, engineer, sales, etc.) or can be defined other ways, such as a full member of a working group or an observer. A role determines what a user sees and can use.

router Intermediary device responsible for deciding which of several paths network (or Internet) traffic will follow.

secret key In public-key cryptography, a private key that is never disclosed to the public. See *public-key cryptography*.

Secure Socket Layer (SSL) A form of secure, low-level encryption that is used by other protocols like HTTP and FTP. The SSL protocol includes provisions for server authentication, encryption of data in transit, and optional client authentication. The version used in i-Planet uses RSA's public and private key encryption, as well as a digital certificate.

self-generated certificate Public key value only used when entities are named using the message digest of their public value, and when these names are securely communicated. See *issued certificate*.

session An i-Planet session is a sequence of interactions between a user and one or more applications, starting with login and ending with logout or timeout.

session key Common cryptographic technique to encrypt each individual conversation between two people with a separate key.

SGML Standard Generalized Markup Language. Method of tagging a document to apply to many format elements.

shared-key cryptography Also known as *symmetric key cryptography*. Cryptography where each party must have the same key to encrypt or decrypt *ciphertext*.

smart card A plastic card with a magnetized strip that is used for authentication.

SMTP Simple Mail Transfer Protocol. Used on the Internet to route email.

SMTP proxy A variant of SMTP that sends messages from one computer to another on a network and is used on the Internet to route email.

SNMP Simple Network Management Protocol. Network management protocol that enables a user to monitor and configure network hosts remotely.

SSL See Secure Socket Layer.

SSL Certificate An electronic token that means you or a vendor have given approval to encrypt and decrypt your secure transactions, using PKI. You create a self-signed SSL Certificate when you install i-Planet software. However, you can also obtain an SSL Certificate from a certificate vendor who authorizes secure communications services over the Internet.

subdomain The next-to-last part of a *fully qualified domain name* that identifies the division or department within a company or organization that own the domain name (for example, eng.sun.com, sales.sun.co.uk); not always specified.

subnet Working scheme that divides a single logical network into smaller physical networks to simplify routing.

subnet mask Specifies which bits of the 32-bit IP address represent network information. The subnet mask, like an IP address, is a 32-bit binary number: a 1 is entered in each position that will be used for network information and a 0 is entered in each position that will be used as node number information. See *node*.

symmetric key cryptography See shared-key cryptography.

TCP See transmission control protocol.

TCP/IP Transmission Control Protocol/Internet Protocol. Protocol suite originally developed for the Internet. It is also called the *Internet* protocol suite. Solaris networks run on TCP/IP by default.

telnet Virtual terminal protocol in the *Internet* suite of protocols. Enables users of one *host* to log in to a remote host and interact as normal terminal users of that host.

telnet proxy An application which sits between the telnet client and telnet server and acts as an intelligent relay.

transmission control protocol (TCP) Major transport protocol in the Internet suite of protocols providing reliable, connection-oriented, full-duplex streams. Uses IP for delivery. Encrypts only IP packet data, but not the headers. Corresponds to the transport layer, which is the fourth of the seven ISO layers. See *TCP/IP*.

transparent clustering A condition whereby multiple machines will appear to the user to be a single machine. In i-Planet, the condition where multiple gateways appear to the user to be a single gateway.

tunneling Process of encrypting an entire IP packet, and wrapping it in another (unencrypted) IP packet. The source and destination addresses on the inner and outer packets may be different.

tunnel address Destination address on the outer (unencrypted) IP packet to which tunnel packets are sent. Generally used for encrypted gateways where the IP address of the host serves as the intermediary for any or all hosts on a network whose topology must remain unknown or hidden from the rest of the world.

URL Uniform Resource Locator. A code that searches for the location of a specific address on the Internet.

user ID Name by which a user is known to the system.

Virtual Private Network A network with the appearance and functionality of a regular network, but which is really like a private network within a public one.

The use of encryption in the lower protocol layers provides a secure connection through an otherwise insecure network, typically the Internet. VPN's are generally cheaper than true private networks using private lines, but rely on having the same encryption system at both ends. The encryption may be performed by firewall software or possibly by routers.

VPN gateway The entry point to a VPN. Typically protected by a firewall.

VPN See *Virtual Private Network*.

WAN Wide area network, a private network (intranet) spanning more than one physical location.

Watchdog A process that monitors a gateway and restarts the gateway if its processes fail.

Web See *World Wide Web*.

Web page Document on the Web.

web server An application that responds to web requests such as HTTP, FTP, etc.

World Wide Web Network of servers on the Internet that provide information and can include hypertext links to other documents on that server and often other servers as well.

Index

A

- APIs, Summary 18
- Audience 13

C

- Command Line Interface 77
 - ipsadmin Command 78
 - Assigning a Role 81
 - Changing a Profile 82
 - Creating New Domain 80
 - Creating New Role 80
 - Deleting a Profile 83
 - Importing New Component 79
 - New User 81
 - Reading a Profile 81
 - Overview 77
 - Sample Code 83
- Content Provider API 51
 - Functionality 51
 - Implementing 53
 - Overview 51
 - Sample Providers 52

G

- glossary 139

H

- HelloServlet Example 87
 - Sample Code
 - Attributes and Privileges 90
 - HelloServlet Properties 88
 - HelloServlet XML 88
 - Prints HTML Output 90
- HelloServletExample
 - Sample Code
 - Initializing the Servlet 91
- HTML Templates 97
 - Authentication Templates 97
 - Customizing Authentication Pages 97
 - HTML Template Files 98
 - Desktop Templates 100
 - Desktop Pages 100
- HTTP/XML Interface 103
 - DTD 104
 - DTD Samples
 - NamingRequest DTD 107
 - NamingResponse DTD 106
 - PLL NotificationSet DTD 106
 - PLL RequestSet DTD 105

- PLL ResonseSet DTD 105
- Log-related DTDs 115
 - Log API Communication DTD 116
- Overview 103
- Profile and Policy-related DTDs 113
 - GetProfile Response XML 115
 - ProfileService DTD 113
- Profile and Policy-related XML 113
 - GetProfile Request XML 115
- Session-Related DTDs 109
 - NotificationSet DTD 109
 - SessionRequest DTD 110
 - SessionResponse DTD 111
- Session-Related XML 109
 - SessionRequest XML 112
 - SessionResponse XML 112
- XML Samples
 - NamingRequest XML 107
 - NamingResponse XML 108

I

- Integrating an Application 23

J

- Javadocs 17

L

- Log API 43
 - Creating Logs 45
 - Deleting Logs 46
 - Functionality 45
 - Implementing 44
 - Log List Retrieval 48
 - Overview 43
 - Querying Logs 48
 - Reading from a Log 47
 - Sample Code 49

- Create a New Log 45
- Delete a Log 46
- Method Handles Logging 95
- Query Log Information 48
- Reading Records from a Log 47
- Retrieve Existing Log List 48
- Sample Log API 49
- Writing Records to a Log 46
- Writing to a Log 46

P

- Pluggable Authentication API 57
 - .properties File 59
 - Overview 57
 - Process 58
 - Writing a Module 62
 - Integration 63
 - Recommendations 63
 - Sample Code 65
- Preface 13
- Profile and Policy API 35
 - Classes and Interfaces 37
 - Dependancies 38
 - Exception Handling 38
 - Functionality 35
 - Importing Classes 40
 - Overview 35
 - Sample Code 41
 - Using 39
- Profile API
 - Sample Code
 - Modify an Attribute 93
 - Policy Checking 94
- Provider Sample 117
 - Attributes and Privileges 119
 - Define the Provider 119
 - Minimal Routines 117
 - Non-Server Functionality 118
 - Requirements and Functionality 118
 - Sample Code
 - XML Defining Attributes and Privileges 120

S

Session API

- Block Diagram 25

- Classes and Interfaces 29

 - Session Class 29

- Implementation 25

- Message Format 28

- Sample Code

 - Get a Session 93

 - HTTP Request and Response 91

 - Session Event 92

- Transport Protocol 28

Single Signon 71

- Instructions for Using 72

- Overview 71

- Sample Code 73

- Special Cases 71

T

Typographic Conventions 15