Oracle® COREid Access and Identity

Developer Guide

10*g* Release 2 (10.1.2)
Part No. B19013-01

May 2005

ORACLE

This program contains third-party code from Apache. Under the terms of the Apache Software License, Oracle is required to provide the following notices. Note, however, that the Oracle program license that accompanied this product determines your right to use the Oracle program, including the Apache software, and the terms contained in the following notices do not change those rights. Notwithstanding anything to the contrary in the Oracle program license, the Apache software is provided by Oracle "AS IS" and without warranty or support of any kind from Oracle or Apache.

* 4. The names "Apache" and "Apache Software Foundation" must
 *   not be used to endorse or promote products derived from this
 *   software without prior written permission. For written
 *   permission, please contact apache@apache.org.
 *
 * 5. Products derived from this software may not be called "Apache",
 *   nor may "Apache" appear in their name, without prior written
 *   permission of the Apache Software Foundation.
 *
 * THIS SOFTWARE IS PROVIDED ``AS IS'' AND ANY EXPRESSED OR IMPLIED
 * WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES
 * OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
 * DISCLAIMED. IN NO EVENT SHALL THE APACHE SOFTWARE FOUNDATION OR
 * ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
 * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
 * USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
 * ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
 * OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT
 * OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 * ==
 *
 * This software consists of voluntary contributions made by many
 * individuals on behalf of the Apache Software Foundation. For more
 * information on the Apache Software Foundation, please see
 * <http://www.apache.org/>.
 *
 * Portions of this software are based upon public domain software
 * originally written at the National Center for Supercomputing Applications,
 * University of Illinois, Urbana-Champaign.
 */

Contents

Preface

The *COREid Developer Guide* explains how to write custom applications and plug-ins that use the programmatic access provided by COREid to gain access to COREid functionality, and, in some cases, to extend that functionality. For more information about what is included in this guide, see “Introduction” on page 17.

Note: Oracle *COREid* was previously known as Oblix *Netpoint*. All legacy references to Oblix and NetPoint, for example, in screen shots, illustrations, and documentation titles, should be understood to refer to Oracle and COREid, respectively.

This Preface covers the following topics:

Intended Audience

This guide is intended for the COREid Administrators assigned during installation and setup, as well as Master Identity Administrators and Delegated Identity Administrators. Administrators configure the rights and tasks available to other administrators and end users.

Information here assumes that you are familiar with your LDAP directory and Web servers, as well as COREid.

COREid Documentation

The manuals that are available for this release include:

Introduction to COREid—Provides an introduction to COREid, a road map to COREid manuals, and a COREid glossary of terms.

COREid Release Notes—Provides up-to-the minute details about the latest COREid release.

COREid Installation Guide—Explains how to install and configure the COREid components.

COREid Upgrade Guide—Explains how to upgrade earlier versions of COREid to the latest version of COREid.

COREid Administration Guide—Explains how to configure COREid applications to display information stored in the directory, how to assign view and modify permissions for data displayed on the COREid applications, and how to assign access controls to users.

COREid Deployment Guide—Provides information for people who plan and manage the environment in which COREid runs. This guide covers capacity planning, system tuning, failover, load balancing, caching, and migration planning.

COREid Customization Guide—Explains how to change the appearance of COREid applications and how to control COREid by making changes to operating systems, Web servers, directory servers, directory content, or by connecting CGI files or JavaScripts to COREid screens. This guide also describes the Access Server API and the Authorization and Authentication Plug-in APIs.

COREid Developer Guide—Explains how to create AccessGates and how to develop plug-ins. This guide also provides information to be aware of when creating CGI files or JavaScripts for COREid.

COREid Integration Guide—Explains how to set up COREid to run with third-party products such as BEA WebLogic, the Plumtree portal, and IBM Websphere.

COREid Schema Description—Provides details about the COREid schema.

Online Help is available from each COREid screen.

Typographical Conventions

COREid manuals use the following typographical conventions:

- When you are instructed to select elements sequentially, the actions are separated with angle brackets, as shown below:

  Click System Admin > System Configuration > View Server Settings.

- Paths to a file are shown using syntax for either the Unix or Windows platform:

  /COREid_install_dir/identity/oblix/logs/debugfile.lst
  COREid_install_dir\identity\oblix\logs\debugfile.lst

  where *COREid_install_dir* refers to the directory where the component, in this case, the COREid Server, is installed.

Contact Information

For a list of contacts including corporate offices world wide, sales, and other details, visit the Oracle Web site at:

`http://www.oracle.com`

You can contact Oracle with questions or comments as follows:

Customer Care—http://www.oracle.com/support/contact.html

Corporate Headquarters

Oracle maintains offices world wide. Oracle corporate headquarters is located at:

500 Oracle Parkway
Redwood Shores, CA 94065
Phone: (650) 506-7000

Before Contacting Customer Care

Before contacting Customer Care, please have available the following:

- Oracle product name and version number
- Type of computer and operating system you are using

Accessing the Customer Care Knowledge Base

For more information about using COREid, see the Oracle Customer Care Knowledge Base. To access the Knowledge Base, you need a login name and password, which you can obtain from your Oracle sales representative.

To access the Knowledge Base:

1. Enter the following URL in your browser and press Return.

   ```
   http://www.oracle.com/support/contact.html
   ```

2. Click the phrase, Login to the Oracle PremiumCare Online Portal.
3. Enter your user name and password in the box that appears, then click Login.
4. Under Oracle Support Tools, click Case Manager.
5. In the next screen, click Find Answers to gain access to the Knowledge Base.

1 Introduction

This guide explains how to write custom applications and plug-ins that use the programmatic access provided by NetPoint to gain access to NetPoint functionality, and, in some cases, to extend that functionality. This guide describes the NetPoint application programming interfaces (APIs) and explains how to:

- Use IdentityXML to interact with NetPoint without using a browser.
- Use the Identity Event API to implement functions and executables triggered by events within NetPoint.
- Use AccessXML to gain access to NetPoint without using a browser.
- Develop custom AccessGates using the Access Server API and the Access Management API in Java, C, C++, and managed code.
- Develop server-side plug-ins to apply custom filters and logic using the Authentication and Authorization plug-in APIs.

This guide includes the following chapters:

2 IdentityXML and NetPoint Web Services

IdentityXML provides a programmatic interface for carrying out the actions that a user can perform when accessing a COREid application from a browser. For instance, a program can send an IdentityXML request to find members of a group defined in the Group Manager application, or to add a user to the User Manager. This chapter describes how to create IdentityXML requests and the process for submitting the requests and handling the responses from NetPoint.

The Web Services Description Language (WSDL) is a schematic description of an XML request. You can use NetPoint's WSDL files as input for generating IdentityXML requests. This chapter describes how you can use NetPoint's WSDL solution as an automated method of generating IdentityXML requests.

Universal Description, Discovery, and Integration (UDDI) is a registry (analogous to the White Pages or Yellow Pages) that enables users to access Web services that are created using WSDL. NetPoint's UDDI and WSDL features together constitute NetPoint's Web Services for Identity Management.

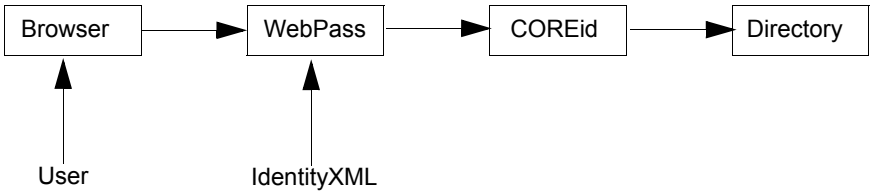
The chapter contains the following sections:

About IdentityXML

IdentityXML provides a programmatic interface for carrying out the actions that a user can perform when accessing a COREid application from a browser. Instead of interacting with the application through a browser, you can write a program. For example, if your company moves and you need to change the area code for the phone number of 100,000 employees, you can use IdentityXML to do a bulk update. Or, if you regularly add employees, instead of doing double entry between your Human Resources application and COREid, you can write a script to call an IdentityXML function to create new users in the User Manager, taking the data from the Human Resources application.

Figure 1 illustrates how IdentityXML works:

Figure 1 IdentityXML Overview



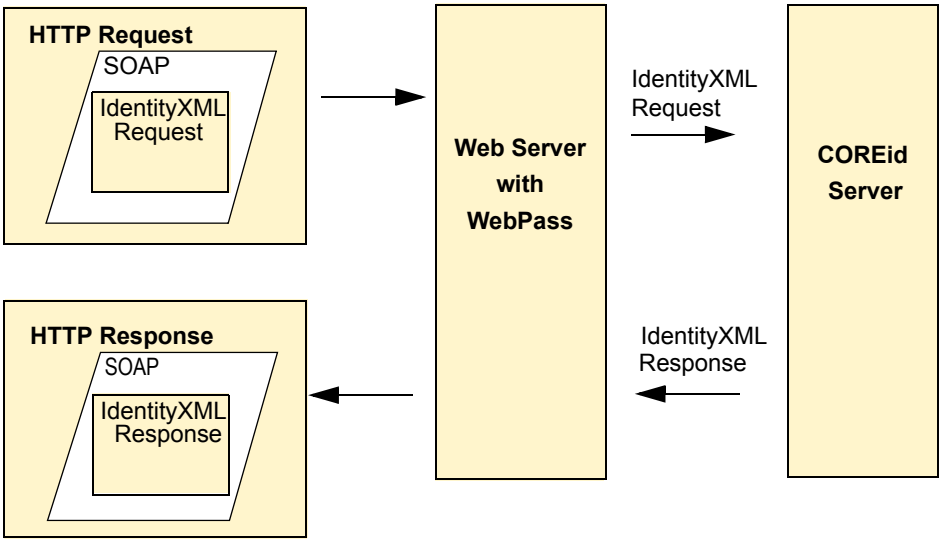
IdentityXML enables you to process simple actions and multi-step workflows to change user, group, and organization object profiles.

IdentityXML allows external applications to access these COREid functions:

- **User**—Create, delete, and manage user data within or outside of a workflow or an asynchronous workflow.
- **Group**—Create, delete, and manage groups and subscriptions.
- **Organization**—Create, delete, and manage organization object data.

To create an IdentityXML request, you look up the request syntax, function names and parameters using the information in this chapter and in “IdentityXML Functions and Parameters” on page 67. After creating the IdentityXML request, you construct a SOAP wrapper to send the IdentityXML request to WebPass using HTTP. Figure 2 illustrates how IdentityXML requests are processed:

Figure 2 IdentityXML Request and Response Flow



IdentityXML requests only work with LDAP attributes that are used on a panel in the User, Group, or Organization Manager.

The IdentityXML API uses XML over SOAP. As shown in "IdentityXML Request and Response Flow" on page 21, you pass IdentityXML parameters to the COREid Server using an HTTP request. This HTTP request contains a SOAP envelope. When WebPass receives the HTTP request, the SOAP envelope indicates that it is an IdentityXML request rather than the usual browser request. The request is forwarded to the COREid Server, where the request is carried out and a response is returned. Alternatively, you can use WSDL to construct the SOAP request.

Data that is sent in a response to an IdentityXML request is similar to the XML output that NetPoint combines with a style sheet to create the HTML that is returned to a browser. You must parse the XML response to extract and use the information you requested.

Note: For a listing of IdentityXML functions and parameters, see "IdentityXML Functions and Parameters" on page 67.

Implementing an IdentityXML Request

Implementing an IdentityXML request requires the procedures identified below.

Task overview: Implementing an IdentityXML Request

1. Decide what COREid operation you want to perform; see the *NetPoint 7.0 Administration Guide Volume 1* for more information.
2. Read "IdentityXML Functions and Parameters" on page 67 to find the function name and parameters that correspond to the operation that you want to perform.
3. Ensure that the IdentityXML request works with LDAP attributes that are configured on a panel in the User, Group, or Organization Manager.

   See the *NetPoint 7.0 Administration Guide* for details.
4. Develop the IdentityXML request and the SOAP envelope for the request, as described in this chapter.
5. Write a program to send an HTTP/S request to NetPoint.

   See sample programs in "Examples" on page 154 and the "SOAP and HTTP Client" on page 667 for details.

   The program can be written in any language. The HTTP/S request must contain an XML payload that consists of the IdentityXML request that you created. You can write a Java program or a Perl script to send the request to a Web server that understands SOAP requests.

   The program or script will do the following:

   a) Identify the host that is responsible for sending the request.

   b) Read in the file that contains the IdentityXML request.

   c) Identify the port to send the data to (port 80).

   d) Identify the cgi that the IdentityXML is being sent to, for example, userservcenter.cgi for the User Manager.

      The cgi files are described in "Locations for Each Application" on page 32.

6. Create a program to parse the XML response and perform any additional processing required.

   For example:

   NetPoint traps the XML request and returns output in the form of an XML document. You need to parse and process this document.

Note: WSDL provides a method for submitting IdentityXML requests through a Java proxy object. This may be more convenient for some developers than the method outlined above. See for “Creating IdentityXML Requests Using WSDL” on page 45 details.

Sending Multiple IdentityXML Requests

Note that each IdentityXML file contains a single request consisting of a single operation. In all likelihood, you will want to use IdentityXML to perform repetitive tasks. For example, suppose that you implement an IdentityXML solution to update an employee’s home address. You may want to re-use this information for subsequent employee address updates. To do this, you need to update the data in the IdentityXML file and resend the request.

You can write a shell or Perl script to dynamically update the data in the IdentityXML request. The script can take information from the original data source and substitute this data in the IdentityXML file that you have set up. This is how, for instance, you could ensure that information about new users entered in your Human Resources database is automatically translated into a Create User operation in NetPoint.

Formatting an IdentityXML Request

All IdentityXML requests use the syntax shown in the following paragraphs. For more information on XML see “XML Background” on page 611. More information on SOAP is provided in “SOAP and HTTP Client” on page 667.

The following listing shows the IdentityXML request format that was used in NetPoint prior to version 6.5.

Listing 1 Pre-6.5 IdentityXML Request Format

```
<?xml version="1.0"?>
<SOAP-ENV:Envelope
     xmlns:oblix="http://www.oblix.com"
     xmlns:SOAP-ENV="http://schemas-xmlsoap.org/soap/envelope/">
<SOAP-ENV:Body>
   <oblix:authentication type="basic">
      <oblix:login>login name</oblix:login>
      <oblix:password>login password</oblix:password>
   </oblix:authentication>
   <oblix:request function="function name"
        application="application name">
      <oblix:params>
         <oblix:param name="param1">value1</oblix:param>
         <oblix:param name="param2">value2</oblix:param>
         <oblix:param name="param3">value3</oblix:param>
      </oblix:params>
   </oblix:request>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Beginning with NetPoint 6.5, there is a new way to specify IdentityXML parameters. The new format is compatible with WSDL and UDDI. See “Creating IdentityXML Requests Using WSDL” on page 45 for details.

The following listing shows the request format that is preferred for NetPoint 6.5 and higher versions:

Listing 2 NetPoint 6.5 and Higher IdentityXML Request Format

```
<?xml version="1.0"?>
<SOAP-ENV:Envelope
     xmlns:oblix="http://www.oblix.com"
     xmlns:SOAP-ENV="http://schemas-xmlsoap.org/soap/envelope/">
<SOAP-ENV:Body>
   <oblix:authentication type="basic">
        <oblix:login>login name</oblix:login>
        <oblix:password>login password</oblix:password>
   </oblix:authentication>
   <oblix:request application="application name"
           function="function name" version="NPWSDL1.0">
      <oblix:params>
           <oblix:param1>value1</oblix:param1>
           <oblix:param2>value2</oblix:param2>
           <oblix:param3>value3</oblix:param3>
      </oblix:params>
   </oblix:request>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

XML Start Tag

This is the start of an XML string:

```
<?xml version="1.0"?>
```

Within this required string you must use a tag to select an encoding specification. for Latin-1 data:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
```

You can use the encoding tag for normal as well as Latin-1 data. Without this encoding string, the default encoding specification is UTF-8.

Soap Tags

The required SOAP tag starts the SOAP root element, the envelope:

```
<SOAP-ENV:Envelope>
   xmlns:oblix="http://www.oblix.com"
   xmlns:SOAP-ENV="http://schemas-xmlsoap.org/soap/envelope/">
```

It is closed by the </SOAP-ENV:Envelope> tag. The namespace attribute xmlns:oblix allows the use of NetPoint-specific tags in the envelope element.

This tag starts the body of the SOAP envelope:

```
<SOAP-ENV:Body>
```

It is closed by the </SOAP-ENV:Body> tag. The body contains two SOAP elements: authentication information and request information.

Authentication Tags

This required element specifies the authentication type to be used:

```
<oblix:authentication type="basic">
```

Currently, basic authentication is the only supported type. This means that the NetPoint login ID and password are needed for authentication. The </oblix:authentication> tag closes this element.

For servers in an Active Directory forest, you need to specify the login domain as well as the login and password. You do this by specifying a <oblix:domain> element within the <oblix:authentication> tag.

An example:

```
<oblix:authentication
      xmlns:oblix="http://www.oblix.com" type="basic">
   <oblix:login>user1k1</oblix:login>
   <oblix:password>abc</oblix:password>
   <oblix:domain>
      DC=locations,DC=oblix,DC=com
   </oblix:domain>
</oblix:authentication>
```

In versions prior to NetPoint 6, this was accomplished by specifying oblix:domain in the parameter section of the request. For example:

```
   <oblix:params>
<oblix:param name="ObLoginDomain">
        DC=locations,DC=oblix,DC=com
</oblix:param>
   </oblix:params>
```

This is still supported.

The login tag:

```
<oblix:login>login name</oblix:login>
```

provides the login ID for a NetPoint user.

The password tag:

```
<oblix:password>login password</oblix:password>
```

provides the actual password of a NetPoint user.

Authentication and SSO Considerations

If your HTTP client can receive and resend the NetPoint SSO cookie, you only need to include the authentication element for the first request in a session. This can reduce the overhead incurred by multiple logins. For an example, see the cookie settings in the sample Java code in "Java Application Example" on page 154. If you submit the SSO cookie as part of the HTTP(S) request, change the IPValidation setting on the WebGate which protects the WebPass that processes the IdentityXML request. Disable IPValidation for the IP address where the request originates. This is usually the Web server hosting the application that submits the IdentityXML request.

There are special considerations if you use both of the following types of request:

- IdentityXML requests that use the SSO cookie on behalf of applications that perform an action for an SSO-authenticated user.
- IdentityXML requests that use Basic authentication for applications that use credentials for privileged operations such as Identity Event API IdentityXML calls.

If your environment supports both types of request, you may require one or more dedicated WebGates and WebPasses for the SSO IdentityXML requests and a separate set of WebGates and WebPasses for the Basic authentication requests.

Request Tag

The request line:

```
<oblix:request application="application name" function="function name" mode =
"modename" version="NPWSDL1.0">
```

tells NetPoint the function to use for the request, for example, search. You replace *function name* with the accurately spelled and capitalized name of the function in double quotation marks. A list of functions starts at "Common Functions" on page 69.

The *application name* can be one of the following:

- **userservcenter**—For User Manager functions.
- **groupservcenter**—For Group Manager functions.
- **objservcenter**—For Organization Manager functions.
- **asynch**—For asynchronous workflows.

You specify the application to send the request to by inputting the correct URL. See "Locations for Each Application" on page 32 and the function descriptions starting with "Common Functions" on page 69 for information on the correct application URL to use with each function.

You can optionally limit the output from this function by providing mode="*modename*" in the request tag. *Modename* takes one of two values.

- **silent**—Returns status information, but no other output. This is useful for IdentityXML functions that test access. The returned status is 0 if the function succeeded, 1 otherwise. To use silent mode, add the following in the line that begins with <oblix:request>:

  ```
  mode="silent"
  ```

  For example:

```
<oblix:request application="userservcenter" function="view" mode="silent">
```

- **dataonly**—Omits display information from the output. The default mode returns all display-related elements in the XML output, including buttons, forms, and so on. Dataonly mode eliminates display-related elements to minimize the size of the output XML.

  For example:

```
<oblix:request application="userservcenter" function="view" mode="dataonly">
```

> **Note:** For the IdentityXML parameter viewGroupMembers, some user interface information is included in the output even in data only mode.

- **version**—The version tag is required if you wish to use the IdentityXML syntax designed for NetPoint 6.5 and higher:

  ```
  version="NPWSDL1.0"
  ```

  Note that the older syntax may be phased out in a future release, so use of version NPWSDL1.0 is recommended.

Parameter Tags

The following:

```
<oblix:params>
```

Delimits a list of *parameter name*:*value* pairs. Note the keyword is params, plural. The tag </oblix:params> closes this element. In NetPoint 6.5 and higher, the params tag may be replaced by other tags, depending on the parameters being invoked. See “Search Parameters” on page 140 and “Attribute Parameters” on page 145 for details.

The pre-NetPoint 6.5 syntax for listing each parameter is as follows:

```
<oblix:param name="param1">value1</oblix:param>
```

Each occurrence of this element provides a specific *parameter name*:*value* pair. You replace *param1* with the parameter name in quotes. Replace *value1* with the actual value. An example:

```
<oblix:param name="uid">
   cn=Marketing Team, ou=Marketing, o=Company, c=US
</oblix:param>
```

Note that this older syntax is supported if you have legacy IdentityXML files.

In NetPoint 6.5 and higher, the preferred method for specifying a parameter is as follows:

```
<oblix:param1>value1</oblix:param1>
```

For example:

```
<oblix:uid>
     cn=Marketing Team, ou=Marketing, o=Company, c=US
</oblix:uid>
```

The NetPoint 6.5 syntax is required for use with the WSDL and UDDI functionality.

Parameters for each function are described starting with “Common Functions” on page 69.

You can supply more than one parameter:value pair:

```
<oblix:param2>value2</oblix:param2>
<oblix:param3>value3</oblix:param3>
```

Request Examples

The following example illustrates an IdentityXML function to change a password. Key words of interest are shown in **bold**:

Listing 3 Sample Change Password Request (pre-NetPoint 6.5 syntax)

```
<?xml version="1.0"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas-xmlsoap.org/soap/envelope/"
xmlns:oblix="http://www.oblix.com">
<SOAP-ENV:Body>
<oblix:authentication xmlns:oblix="http://www.oblix.com" type="basic">
     <oblix:login>dadmin</oblix:login>
     <oblix:password>password</oblix:password>
</oblix:authentication>
<oblix:request application="userservcenter" function="modifyUser"
   mode="" version="NPWSDL1.0"> <oblix:params>
   <oblix:param name="uid">uid=jones,ou=People,ou=NA,ou=DEALER,dc=company,dc=com
   </oblix:param>
   <oblix:param name="attrName_1">userPassword</oblix:param>
   <oblix:param name="attrValue_1">password</oblix:param>
```

```
   <oblix:param name="attrValue_1_confirm">password</oblix:param>
   <oblix:param name="attrValue_1_old">d</oblix:param>
   <oblix:param name="attrOperation_1">REPLACE</oblix:param>
   <oblix:param name="noOfFields">1</oblix:param>
</oblix:params>
</oblix:request>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Listing 4 Sample Change Password Request (NetPoint 6.5 and higher syntax)

```
<?xml version="1.0"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas-xmlsoap.org/soap/envelope/"
xmlns:oblix="http://www.oblix.com">
<SOAP-ENV:Body>
<oblix:authentication xmlns:oblix="http://www.oblix.com" type="basic">
      <oblix:login>dadmin</oblix:login>
      <oblix:password>password</oblix:password>
</oblix:authentication>
<oblix:request application="userservcenter" function="modifyUser" mode=""
version="NPWSDL1.0">
<oblix:attributeParams>
   <oblix:uid>uid=jones,ou=People,ou=NA,ou=DEALER,dc=company,dc=com</oblix:uid>
   <oblix:PasswordAttribute>
      <oblix:attrName>userPassword</oblix:attrName>
      <oblix:attrNewValue>password</oblix:attrNewValue>
      <oblix:attrConfirmValue>password</oblix:attrConfirmValue>
      <oblix:attrOldValue>d</oblix:attrOldValue>
      <oblix:attrOperation>REPLACE</oblix:attrOperation>
      <oblix:attrNoOfFields">1</oblix:attrNoOfFields>
   </oblix:PasswordAttribute>
</oblix:attributeParams>
</oblix:request>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

In the examples above:

- **modifyUser**—This is the name of an IdentityXML function. This function changes a user attribute in the User Manager.
- **oblix:authentication**—This is the authentication tag that allows the user to log in.
- **oblix:params (pre-NetPoint 6.5 syntax) or oblix:attributeParams (NetPoint 6.5 and higher syntax)**—The uid identifies the user whose password is to be changed.
- **attrName**—This identifies the names of one or more attributes to be viewed or changed.

- **attrValue_1 (pre-NetPoint 6.5 syntax) or attrNewValue (NetPoint 6.5 and higher syntax)**—This identifies the value that is to be provided for the attribute identified by the attrName parameter.

Listing 5: "Sample IdentityXML Request (pre-NetPoint 6.5 syntax)" on page 31 and Listing 6: "Sample IdentityXML Request (NetPoint 6.5 and higher syntax)" on page 31 show an IdentityXML function that performs a query. This query asks if the logged in user has permission to view a particular group profile. This request might be sent to the User Manager at the following URL:

```
http://www.customer.com/identity/oblix/apps/userservcenter/bin/
userservcenter.cgi
```

NetPoint first authenticates John Smith as a valid user, and verifies that the user is authorized to do a password change. NetPoint searches the User Manager for all entries under the Employees tab that have john as a substring match in their cn attribute. Because mode="silent" is part of the request, the response only contains status information.

Listing 5 Sample IdentityXML Request (pre-NetPoint 6.5 syntax)

```
<?xml version="1.0"?>
<SOAP-ENV:Envelope
      xmlns:oblix="http://www.oblix.com"
      xmlns:SOAP-ENV="http://schemas-xmlsoap.org/soap/envelope/">
<SOAP-ENV:Body>
   <oblix:authentication type="basic">
      <oblix:login>J.Smith</oblix:login>
      <oblix:password>J.Smith</oblix:password>
   </oblix:authentication>
   <oblix:request function="search" mode="silent">
      <oblix:params>
         <oblix:param name="tab_id">Employees</tab_id>
         <oblix:param name="STy1">cn</oblix:param>
         <oblix:param name="SLk1">OSM</oblix:param>
         <oblix:param name="SSt1">john</oblix:param>
      </oblix:params>
   </oblix:request>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Listing 6 Sample IdentityXML Request (NetPoint 6.5 and higher syntax)

```
<?xml version="1.0"?>
<SOAP-ENV:Envelope
. . .
   </oblix:authentication>
   <oblix:request function="search" mode="silent"
      version="NPWSDL1.0">
    <oblix:AttributeParams>
        <oblix:tab_id>Employees</oblix:tab_id>
      <oblix:SearchParams>
```

```
      <oblix:Condition>
         <oblix:SearchAttr>cn</oblix:SearchAttr>
         <oblix:SearchOperation>OSM</oblix:SearchOperation>
         <oblix:SearchString>john</oblix:SearchString>
      </oblix:Condition>
      </oblix:SearchParams>
   </oblix:AttributeParams>
   </oblix:request>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Important: As shown in Listing 6: "Sample IdentityXML Request (NetPoint 6.5 and higher syntax)" on page 31, you must set the version string to NPWSDL1.0. For example, myrequest.setVersion("NPWSDL1.0");.

Handling Special Characters in Requests

In an XML document, if you want special characters, such as the angle bracket "<" to be treated as text, they must be encoded. The following table summarizes the handling of special characters in XML:

Special character	Description	Encoding
>	Begins a tag.	>
<	Ends a tag.	<
"	Quotation mark.	"
'	Apostrophe.	'
&	Ampersand.	&

Locations for Each Application

The applications that respond to IdentityXML input and the files that they use are as follows.

URLs to the applications are as follows:

- **For the Group Manager**—http://www.customer.com:*port*/identity/oblix/apps/groupservcenter/bin/groupservcenter.cgi
- **For the Organization Manager**—http://www.customer.com:*port*/identity/oblix/apps/objservcenter/bin/objservcenter.cgi
- **For the User Manager**—http://www.customer.com:*port*/identity/oblix/apps/userservcenter/bin/userservcenter.cgi

- **For Asynchronous Workflows**—http://www.customer.com:*port*/identity/oblix/apps/asynch/bin/asynch.cgi

The schema files are as follows:

- **XML schema documentation files**—*WebPass_install_dir*\oblix\WebServices\XMLSchema*.xsd
- **WSDL schema files**—*WebPass_install_dir*\oblix\WebServices\WSDL*.wsdl
- **UDDI sample Java files**—*WebPass_install_dir*\oblix\WebServices\UDDI*.*

The style sheets are as follows:

- **Group Manager**—*COREid_install_dir*/apps/groupservercenter/ui/style0
- **Organization Manager**—*COREid_install_dir*/apps/objservercenter/ui/style0
- **User Manager**—*COREid_install_dir*/apps/userservercenter/ui/style0
- **Asynchronous Workflows**—none

Types of IdentityXML Functions

There are three types of IdentityXML functions:

- **Test**—These functions test whether the user is allowed to perform a particular function. Test functions can be used before doing large scale batch operations. Test functions return a yes or no type of response.
- **Get**—These functions show current directory content.
- **Set**—These functions change current directory content.

All functions are listed in “IdentityXML Functions and Parameters” on page 67. Note that parameters for these functions can be specified in any order. You do not need to follow the order provided in the parameter descriptions.

Functions to Test Access to Data

Use IdentityXML test functions to determine if you or another user can perform a specific function. Functions that begin with CanI are a direct (first-person) test. Functions that begin with CanUser are an indirect (third-person) test. These functions ask "may user J. Smith do something." A third person test is also called a *proxy test*. You identify the person who is the target of the test using the proxysourceuid parameter.

The following is an example test request.

Listing 7 Test Request Example (NetPoint 6.5 and higher syntax)

```
<?xml version="1.0"?>
<SOAP-ENV:Envelope
      xmlns:oblix="http://www.oblix.com"
      xmlns:SOAP-ENV="http:
                  //schemas-xmlsoap.org/soap/envelope/">
   <SOAP-ENV:Body>
      <oblix:authentication type="basic">
         <oblix:login>J.Smith</oblix:login>
         <oblix:password>J.Smith</oblix:password>
      </oblix:authentication>
      <oblix:request function="canIViewGroupProfile"
         version="NPWSDL1.0">
         <oblix:AttrParams>
            <oblix:uid>
               cn=Marketing Team,ou=Marketing,o=Company,c=US
            </oblix:uid>
         </oblix:AttrParams>
      </oblix:request>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The result of the request appears as the value in an ObTextMessage element, within the ObAccessAPIResult element. There are three possible results.

- **Allowed**—You or the specified user may do the requested activity.
- **Denied**—You or the specified user may not do the requested activity.
- **Not authorized to use service**—You lack the rights necessary to make the request, as described in "Privileges to View and Modify" on page 37.

The following is an example test response.

Listing 8 Test Response Example

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<SOAP-ENV:Envelope
      xmlns:oblix="http://www.oblix.com"
      xmlns:SOAP-ENV="http://
         schemas-xmlsoap.org/soap/envelope/">
   <SOAP-ENV:Body>
      <Oblix>
         <ObAccessAPIResult>
            <ObRequestInfo>187658080</ObRequestInfo>
            <ObTextMessage>Allowed</ObTextMessage>
         </ObAccessAPIResult>
      </Oblix>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Functions to Get Data

Some IdentityXML functions gather and return information from the directory. For functions that get data for a logged in user, the user must have view privileges for the target object naming attribute and the specified attribute.

The following is an example of a request for workflow ticket information.

Listing 9 Request for Workflow Ticket Information

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://
      schemas-xmlsoap.org/soap/envelope/">
   <SOAP-ENV:Body>
      <oblix:authentication xmlns:oblix="http://
            www.oblix.com" type="basic">
         <oblix:login>J.Smith</oblix:login>
         <oblix:password>J.Smith</oblix:password>
      </oblix:authentication>
      <oblix:request function="workflowTicketInfo"
         version="NPWSDL1.0">
         <oblix:AttrParams>
            <oblix:workflowInstanceDn>
                  obwfinstanceid=20001019T1609090,
                  obcontainerId=workflowInstances,
                  o=Oblix,o=Company,c=US
            </oblix:workflowInstanceDn>
            <oblix:workflowStepInstanceId>
                  2
            </oblix:workflowStepInstanceId>
         <oblix:AttrParams>
      </oblix:request>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Functions to Set Data

These functions change directory content. Listing 10: "Subscription to a Group" on page 37 is an example of subscribing to a group.

For functions that allow a user to set data for themselves:

- The user must have view privileges for the target object naming attribute.
- For a workflow or a request to set an attribute, the logged in user must have view privileges for the target object naming attribute and the attribute requested to be set, and the user must be a participant of the appropriate workflow.
- For a workflow or request to delete a user, group, or object, the logged in user must have view access to the target object naming attribute and be a participant of the appropriate workflow.
- For a workflow or request to create a user, group or object, the searchbase rule does not apply. If a domain is specified the logged in user must be a participant of the matching workflows for that target domain. If no domain is specified, the logged in user must be a participant of any matching workflows.

For functions that allow a logged in user to set data for another user:

- All of the above privileges must apply to the proxysourceuid (that is, the user in the “CanUser. . .” call).
- The logged in user must have view privileges for the class attribute of the proxysourceuid and the targetuid if it exists. For example, a CanUserView type of call has a targetuid but a CanUserCreate call does not.
- The logged in user must have grant and read privileges for the class attribute of the proxysourceuid and the targetuid if one exists.

For common IdentityXML functions and application specific IdentityXML functions:

- All the applications should have the same access privileges as the equivalent GUI function.
- **Exceptions**: the rules that apply to the indirect access functions above also apply to the following group functions: userGroupsProfile, subscribeUserToGroup, unsubscribeUserFromGroup.

Listing 10 Subscription to a Group

```
<?xml version="1.0"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://
      schemas-xmlsoap.org/soap/envelope/">
   <SOAP-ENV:Body>
      <oblix:authentication xmlns:oblix="http://
            www.oblix.com" type="basic">
         <oblix:login>J.Smith</oblix:login>
         <oblix:password>J.Smith</oblix:password>
      </oblix:authentication>
      <oblix:request function="subscribeToGroup"
         version="NPWSDL1.0">
         <oblix:params>
            <oblix:uid>
               cn=Marketing Team, ou=Marketing,
               o=Company, c=US
            </oblix:uid>
         </oblix:params>
      </oblix:request>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Privileges to View and Modify

You use IdentityXML requests to look at or change data in the directory. The ability to view or change data is controlled by the view, modify, and grant rights that a NetPoint Administrator assigns to you. See the *NetPoint 7.0 Administration Guide Volume 1* for details.

For most functions, except where noted, the data you are attempting to view or change must be in the searchbase that the administrator set for you. For example, if your searchbase is limited to the Sales organization, you cannot view or change data in the Accounting organization.

Privileges Required for Direct Access APIs

Direct functions test your own ability to view or change data.

For functions that test your ability to view a value without using a workflow:

- You must have view privileges for the target object naming attribute
- If an attribute is specified, you must have permission to view the attribute
- The attribute must be included in a panel configured for a COREid application.

For functions that test your ability to modify a value without using a workflow:

- You must have view privileges for the target object naming attribute.

- You must have write privileges for the target attribute to be set.
- The attribute must be on a panel configured for a COREid application.

For functions that use a workflow:

- To test for the ability to modify attributes, you must:
  - Have view permissions for the target object naming attribute (for example, the uid or tab_id).
  - Have view permissions for the target attribute.
  - Be a participant in the workflow that is used to set that attribute.
- To test for the ability to delete, you must:
  - Have view permissions for the target object naming attribute.
  - Be a participant in the workflow that is used to delete the object.
- To test for the ability to create, you must:
  - If a domain is specified—Be a participant in the workflow that is used to create the data in that domain.
  - If a domain is not specified—Be a participant of at least one workflow that creates that data.

Note: Workflow governs in all three categories. For the create test, if you are a participant in the workflow, you will be granted access even if the object is outside of your assigned searchbase. For all of the tests, if you are not a participant in the workflow, you will get a negative response even if you have modify rights to the attribute.

Privileges Required for Indirect Access APIs

Indirect functions test the ability of another user, represented by the proxysourceuid parameter, to view data or make changes. This parameter is required for a number of IdentityXML functions, as described in “IdentityXML Functions and Parameters” on page 67. Required privileges are as follows:

- All the access privileges described above must be satisfied for the person represented by the proxysourceuid parameter.
- You must have view privileges for the class attribute of the proxysourceuid and the targetuid (if used).
- The object classes for the proxysourceuid and targetuid must be in your searchbase.
- You must have the ability to grant the right to read on the class attribute of the proxysourceuid and the targetuid (if used).

Privileges Required for Application-Specific IdentityXML Requests

Application-specific IdentityXML requests are the get or set functions that view or change data. Each is equivalent to an operation that can be carried out through the GUI, and the rights are those that would apply to the GUI.

Exceptions are the three functions listed below. Rights for these must be the same as for the Indirect Access APIs.

```
userGroupsProfile

subscribeUsertoGroup

unsubscribeUserfromGroup
```

Note: In any IdentityXML request, the LDAP attributes that can be specified or used are only those that have been configured in the COREid System and are part of a panel in the profile of the user, group, or organization. All other attributes are considered invalid.

Privileges Required for DN Operations

Some parameters take values of type DN. Privileges required for DN operations are as follows:

- **View**—If you submit a request to view a DN attribute value (for example, by using the attrName function), only values for which you have view permissions and localized permissions are returned. That is, you must have read access to the class attribute of that DN, and the DN value should fall under your searchbases with respect to the type of its object class.
- **Modify**—If you submit a request to add, modify, or delete a DN attribute value (for example, through any modify or workflow function), values are considered valid only if you have view permissions and localized permissions for them. That is, you should have read access to the class attribute of that DN, and that DN value should fall under your searchbases with respect to the type of its object class. If you specify an invalid DN value, an error message such as “Invalid value for parameter uniqueMember” is returned.

  Some examples of invalid DN values are junk values, deactivated users, or DNs that do not satisfy your access rights.

Formatting an IdentityXML Response

The chapter on PresentationXML in the *NetPoint 7.0 Customization Guide* discusses the way the HTML response is built up. See “XML Background” on page 611 for a discussion of XSD and XML content.

Depending upon the the COREid application being used, you locate the matching XML registration file (userservcenterreg.xml for example). Within the registration file, look for the following element:

```
ObProgram name="xxxxxx"
```

where *xxxxx* is the function you are using. In this example, you look for ObProgram name=”search”. Within that element is another:

```
ObSchema name="yyyy"
```

where *yyyy* is the name of the XML schema file that defines the expected output. In this example, that line reads as follows:

```
ObSchema name="usc_search.xsd"
```

The XML schema file generally begins with several includes, but the output XML starts with the first element which contains a reference to ObRequestInfo, and will contain only the information specified by that element.

For example, within the usc_search.xsd file the element ObSearch contains the ObRequestInfo element, as shown in Listing 11: "Response Format" on page 40, taken from that file.

Listing 11 Response Format

```
<xsd:element name="ObSearch">
   <xsd:complexType>
      <xsd:sequence>
         <xsd:element ref="ObRequestInfo"/>
         <xsd:element ref="ObScripts"/>
         <xsd:element ref="ObForm"/>
         <xsd:element ref="ObTextMessage"/>
         <xsd:element ref="ObColumnInfo"/>
         <xsd:element ref="ObEntry" maxOccurs="unbounded"/>
         <xsd:element ref="ObButton" maxOccurs="unbounded"/>
         <xsd:element
               ref="ObViewModeButtonsForSearchResults"/>
         <xsd:element ref="ObStatus"/>
      </xsd:sequence>
   </xsd:complexType>
</xsd:element>
```

Detailed search results are returned within ObAttribute elements nested within an ObEntry element. An ObStatus element returns the status value for the request:

- An ObStatus value of 0 means the request was accepted and processed.
- A value of 1 means that an error has occurred.

The recommended strategy for working with the response data is to use a tool, such as the HTTPClient discussed in “SOAP and HTTP Client” on page 667, to get a sample of the output returned by NetPoint. With the corresponding XML schema as a guide, you can determine which parts of the data you want your application to use.

Parsing a Response

IdentityXML responses adhere to a particular XML schema. Due to the nature of attribute mapping in the NetPont COREid System, an attribute can be configured as one of many possible data types, for instance, as a single-valued string, a multi-valued string, various date formats, integers, selection lists, checkboxes, and so on. As a result, Oblix does not recommend hard-coding the attribute-to-data-type parsing dependencies. It is recommended that you implement a parser that can recognize the data type and extract the relevant data and attribute properties.

The IdentityXML response structure follows the data definition for a particular object class type. For example, a profile for an object such as user, group, or organization consists of at least one panel of attributes. An attribute may appear in more than one panel in the COREid application. The order of the attributes is determined by configuration settings. It is a common mistake in IdentityXML implementations to make invalid assumptions such as the number of occurrences of an attribute in an XML response or that an attribute will always have a value.

Response Example

Listing 12: "Response Example." on page 42 is an actual response to the example search request. There would be an ObEntry element returned for each directory entry satisfying the search.

Listing 12 Response Example.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://
      schemas-xmlsoap.org/soap/envelope/">
<SOAP-ENV:Body>
<Oblix>
<ObSearch>
<ObRequestInfo>181481520</ObRequestInfo>
<ObScripts>
...
</ObScripts>
<ObForm ....>
...
</ObForm>
<ObTextMessage/>
<ObColumnInfo>
...
</ObColumnInfo>
<ObEntry>
   <ObAttribute obattrName="cn">
      <ObDisplay obdisplayName="Name" obdisplayType="dn"
           obname="cn" obmode="view" obcanRequest="false"
           obrequired="false">
         <ObDn>
            <ObLink obdisplayName="John Fulton"
                 obhref="userservcenter.cgi
                 ?program=view&tab_id=Employees
                 &uid=cn%3DJohn%20Fulton%2C
                 %20ou%3DEngineering
                 %2C%20o%3DCompany%2C%20c%3DUS"
                 obmouseOver="View personal information">
                 cn=John Fulton, ou=Engineering,
                      o=Company, c=US
               <ObImage obhref="CIMAGEperson"
                    obalt="View personal information" />
            </ObLink>
         </ObDn>
      </ObDisplay>
   </ObAttribute>
```

Listing 12 (Continued) Response Example.

```
      <ObAttribute obattrName="mail">
      <ObDisplay obdisplayName="E-Mail Address"
            obdisplayType="email" obsemanticType="ObSEmail"
            obname="mail" obmode="view" obcanRequest="false"
            obrequired="false">
         <ObEmail>
            <ObValue>J.Fulton@company.com</ObValue>
         </ObEmail>
      </ObDisplay>
   </ObAttribute>
. . .
   <ObAttribute obattrName="telephonenumber">
      <ObDisplay obdisplayName="Phone Number"
            obdisplayType="textS”obname="telephonenumber"
            obmode="view" obcanRequest="false"
            obrequired="false">
         <ObTextS>
            <ObValue>408-555-1173</ObValue>
         </ObTextS>
      </ObDisplay>
   </ObAttribute>
   <ObAttribute obattrName="ou">
      <ObDisplay obdisplayName="Organization"
            obdisplayType="select" obname="ou"
            obmode="view" obcanRequest="false"
            obrequired="false">
         <ObSelect>
            <ObChoice obdisplayName="Engineering"
               obselected="true">Engineering
            </ObChoice>
         </ObSelect>
      </ObDisplay>
   </ObAttribute>
</ObEntry>
...
<ObViewModeButtonsForSearchResults>
...
</ObViewModeButtonsForSearchResults>
<ObStatus>0</ObStatus>
</ObSearch>
<ObStatus>0</ObStatus>
</Oblix>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Error Responses

If a request contains invalid data, or if you try to access data for which you have no authorization, you will get an error. The error response shown here is the result of using *XXX* as the value for the SLk1 parameter in the request. Note that the response includes the element ObError and the element ObStatus with the value 1, at the same indent level as ObError. Look for both of these parameters to identify error responses.

Listing 13 Error Response

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<SOAP-ENV:Envelopexmlns:SOAP-ENV="http://
      schemas-xmlsoap.org/soap/envelope/">
   <SOAP-ENV:Body>
      <Oblix xmlns:oblix="http://www.oblix.com/"
            xmlns="http://www.oblix.com/">
         <ObError>
            <ObRequestInfo>187658080</ObRequestInfo>
            <ObTextMessage>
            The attribute specified for this
                  search (XXX) is either not searchable
                  or not a valid attribute.
            </ObTextMessage>
            <ObStatus>1</ObStatus>
         </ObError>
         <ObStatus>1</ObStatus>
      </Oblix>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The following are some possible errors:

1. Invalid parameter value: %1

   This is returned when an input parameter has an invalid value. This could be because the parameter is not provided in a DN format, or does not exist in the schema. %1 is replaced with the name of the parameter that was in error, for example ObWorkflowName.

2. Invalid parameters.

   This is returned when the required or optional attributes provided for a workflow are not valid; for example, if the password is set to a minimum of eight characters and the input is only three characters.

3. You do not have access rights.

   You do not have the right to perform the operation

4. There is an XML syntax error. That is, there is an error in the code, such as a typo.

5. There is no profile configured for this kind of user.

   This is a generic error generated when the input is invalid and is not caught by other error catching.

6. A value is required for %1.

   This error indicates that a required parameter is missing, perhaps for a workflow attribute or as part of a delete request.

7. Not authorized to use service

   You have not been authenticated, or lack the authorization, to make requests to particular application.

Creating IdentityXML Requests Using WSDL

A Web Service consists of programmable application logic that is accessed using standard Internet protocols. XML Web Services expose useful functionality to Web users through a standard Web protocol. In most cases, the protocol used is SOAP. XML Web Services provide a way to describe an interface in enough detail to allow a developer to build a client application to talk to it. The description of the interface is usually provided in an XML document called a Web Services Description Language (WSDL) document.

WSDL provides a convenient method for working with Web requests that are created in XML. A WSDL file is a schematic description of an XML request. The contents of a WSDL file consists of information about an XML function name, its parameters, and so on.

The following sections describe using NetPoint's WSDL files and working with them as an alternative method for generating IdentityXML requests.

Note: WSDL and UDDI are additions to the native IdentityXML functionality. If you prefer to work directly with IdentityXML files, this is still supported.

WSDL is part of the Microsoft .NET framework, but you do not need to use the rest of the framework to use WSDL.

Benefits of WSDL

WSDL enables you to create services that can anyone can access on the Web. This allows others to build new, more powerful applications that use XML Web services as building blocks. The section “Making WSDL Functions Available Using UDDI” on page 65 describes how to register WSDL functions so that they are available to anyone who needs to use them.

WSDL also provides an abstraction layer for IdentityXML. If you rely on IdentityXML for integration with Web application servers or third-party applications, or if you work with a variety of application frameworks and separate development teams, it is unlikely that all of the application developers would have expertise in IdentityXML. WSDL provides tools that allow you to bypass directly coding IdentityXML calls. Developers can use tools to generate proxy code for the IdentityXML function, and use the proxy code to make the calls. This allows the developer to use WSDL to avoid hands-on XML programming.

As noted in “About IdentityXML” on page 20, you hand-craft an IdentityXML request document by looking up the request syntax, function names and parameters in this guide, constructing an XML-based SOAP request, and sending the IdentityXML request to the WebPass using HTTP. With WSDL, you only work with objects, rather than hand-crafting the XML request. Using WSDL, the code for sending the request is generated automatically. You only need to set the parameters in the request, rather than constructing the entire request.

About NetPoint WSDL Files

NetPoint provides WSDL files for each of the IdentityXML functions described in “IdentityXML Functions and Parameters” on page 67. The WSDL files are in the following location:

oblix\WebServices\WSDL*.wsdl

The file names reflect the name of the function, for instance, one WSDL file contains the name “search” because it corresponds to the IdentityXML search function. Another WSDL file contains the name workflowTicketSearch, which corresponds to another IdentityXML function. For a complete list of function names, see “IdentityXML Functions and Parameters” on page 67.

WSDL Directory Structure

The directory oblix\WebServices is structured as follows:

- **WSDL**—Contains WSDL template files, each of which corresponds to an IdentityXML function.
- **XMLSchema**—Contains schema required for generating proxy objects.
- **Samples**—Contains the following:

- **UDDI**—Contains sample files for implementing UDDI functions. See "Making WSDL Functions Available Using UDDI" on page 65 for details.
- **WSDL**—Contains sample code for invoking Web services using Java and .NET.

In the directory oblix\WebServices\WSDL*.wsdl, WSDL files are named as follows:

- **common_*.wsdl**—Each file contains the information required for generating a Common IdentityXML request.
- **gm_*.wsdl**—Each file contains the information required for generating a Group Manager IdentityXML request.
- **um_*.wsdl**—Each file contains the information required for generating a User Manager IdentityXML request.
- **om_*.wsdl**—Each file contains the information required for generating an Organization Manager IdentityXML request.

The Oblix implementation of WSDL follows the recommended model for publishing into UDDI. This model calls for two files to be present for each function:

- There is one file for each IdentityXML function that contains the URL location of the function. The name of this WSDL file contains the IdentityXML function name, with a prefix of "common_", "gm_", "um_", or "om_". For example, the search function is a common function, so the corresponding WSDL file is called common_search.wsdl. The function to view a group profile is called view, so the corresponding WSDL file is called gm_view.wsdl.
- There is a second WSDL file for the function interface. This file always contains the string "interface" in the file name.

WSDL Documents

A WSDL document has two main sections. The first section consists of abstract definitions. These are provided in the NetPoint-supplied WSDL documents:

- **Types**—Machine- and language-independent type definitions.
- **Messages**—These contain function parameters.
- **PortTypes**—These contain descriptions of function components (operation name, input parameters, and output parameters).

The second section consists of concrete definitions. This information is specific to your environment:

- **Bindings**—The binding(s) of each operation in the PortTypes section.

- **Services**—The port address(es) for each binding.

Each WSDL file imports another WSDL file of the same name plus a suffix of "_interface." For example, gm_view.wsdl file imports a file called gm_view_interface.wsdl. The interface WSDL file contains the attribute types, function name, binding and so on. This file is the abstract representation.

The file that corresponds to the name of the IdentityXML function contains the implementation definition. It contains the URL where this Web service can be invoked. This is the URL to the NetPoint installation. This file imports the file with the name that is a concatenation of the name of this file and "_interface", for example gm_view_interface.wsdl.

Providing two WSDL files for each IdentityXML function is helpful if you need multiple implementations of the same interface. You can expose the interface files once through UDDI, and the multiple implementation files can also be published through UDDI.

Sample WSDL Files

The following are examples of an actual NetPoint WSDL document and a second WSDL file that is included in the first file. Note that the function name is shown in **bold**.

The example below shows the WSDL document that corresponds to the IdentityXML search function:

Listing 14 Common_search.wsdl file

```
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap"
   xmlns:http="http://schemas.xmlsoap.org/wsdl/http"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
   xmlns:oblix="http:www.oblix.com"
   xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
   xmlns:tns="http://www.oblix.com/wsdl/common_search"
   targetNamespace="http://www.oblix.com/wsdl/common_search">
<import namespace="http://www.oblix.com/"
   location="common_search_interface.wsdl"/>
<service name="OblixIDXML_common_search_Service">
<port name="OblixIDXML_common_search_Port"
   binding="tns:OblixIDXML_common_search_Binding">
<soap:address location ="http://echo.oblix.com:5555/identity/oblix/apps/
userservcenter/bin/userservcenter.cgi"/>
</port>
</service>
</definitions>
```

The following example shows the interface file that provides many of the definitions used in the common_search.wsdl file, above:

Listing 15 Common_search_interface.wsdl

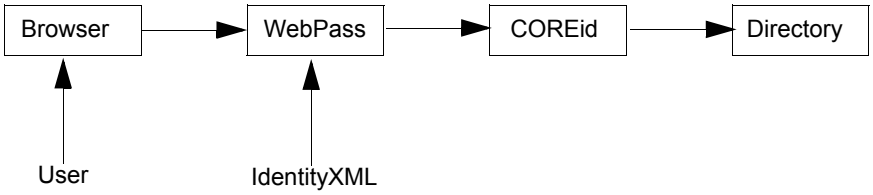
```
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
   . . .
   xmlns:oblix="http://www.oblix.com/"
   xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
   xmlns:tns="http://www.oblix.com/wsdl/common_search"
      targetNamespace="http://www.oblix.com/wsdl/common_search">
<import namespace="http://www.oblix.com/" location="../XMLSchema/
common_parameters.xsd"/>
<import namespace="http://www.oblix.com/" location="../XMLSchema/
common_authentication.xsd"/>
<import namespace="http://www.oblix.com/" location="../XMLSchema/
common_component_search.xsd"/>
<types>
<xsd:schema targetNamespace="http://www.oblix.com/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<xsd:element name="request">
<xsd:complexType>
      <xsd:sequence>
      <xsd:element name="params">
         <xsd:complexType>
         <xsd:sequence>
            <xsd:element ref="oblix:tab_id" minOccurs="0"/>
            <xsd:element ref="oblix:startFrom" minOccurs="0"/>
            <xsd:element ref="oblix:noOfRecords" minOccurs="0"/>
            <xsd:element ref="oblix:noOfFields" minOccurs="0"/>
            <xsd:element ref="oblix:showAllResults" minOccurs="0"/>
            <xsd:element ref="oblix:sortBy" minOccurs="0"/>
            <xsd:element ref="oblix:sortOrder" minOccurs="0"/>
            <xsd:element ref="oblix:attrName" minOccurs="0"
               maxOccurs="unbounded"/>
      . . .
      <!--All of these functions can be invoked for any COREid application -->
      <!--User Manager, Group Manager, or Organization Manager to get the -->
      <!--right search results. They are described in this one WSDL file. -->
      <xsd:pattern value="userservcenter\groupservcenter\objservcenter"/>
      </xsd:restriction>
      </xsd:simpleType>
      </xsd:attribute>
   <xsd:attribute name="function" type="xsd:string" use="required"/>
   <xsd:attribute name="mode" type="xsd:string" use="optional"/>
</xsd:complexType>
</xsd:element>
</xsd:schema>
</types>
<message name="OlibxIDXMLInput">
<part name="authentication" element="oblix:authentication"/>
<part name="request" element="oblix:request"/>
```

```
</message>
<message name="OblixIDXMLOutput">
<part name="body" element="oblix:Oblix"/>
</message>
<portType name="OblixIDXMLPortType">
   <operation name="OblixIDXML_common_search">
   <input message="tns:OblixIDXMLInput"/>
   <output message="tns:OblixIDXMLOutput"/>
   </operation>
   <portType>
<binding name="OblixIDXML_common_search_Binding" type="tns:OblixIDXMLPortType">
<soap:binding style="document"
   transport="http://schemas:xmlsoap.org/soap/http"/>
<operation name="OblixIDXML_common_search">
<soap:operation soapAction="http://www.oblix.com/"/>
...
</definitions>
```

About Working With WSDL Files

As illustrated in Figure 3, IdentityXML calls can substitute for user interaction with the COREid System:

Figure 3 IdentityXML can Substitute for User Interaction



You can either construct IdentityXML requests and SOAP envelopes manually, or you can use WSDL to automatically generate a client object. You then only need to edit the client object to set the appropriate parameters.

Task overview: Working with the NetPoint WSDL files

1. Edit the appropriate WSDL files.
2. Generate a Java or .NET proxy object.
3. Develop a Java or .NET client.

The following sections provide details on how to develop a Java or .NET WSDL solution.

If you are familiar with Java programming, NetPoint's Web Services for Identity Management enables you to use WSDL instead of working directly with IdentityXML. NetPoint provides two WSDL files for each IdentityXML function. You use these files to generate a Java proxy object for your IdentityXML requests.

Task overview: Using WSDL to generate Java IdentityXML requests

1. Identify the IdentityXML request you want to generate.
2. Look up the function that you wish to use in “IdentityXML Functions and Parameters” on page 67.

   Optionally, you can locate the corresponding WSDL file in a UDDI registry. NetPoint provides WSDL files in a local installation directory. However, if you have access to a UDDI registry containing the WSDL function, this can be a convenient method of locating the function. See “Making WSDL Functions Available Using UDDI” on page 65 for details.
3. Edit the host name and port number in the soap:address statement in a corresponding WSDL document.

   For example, for Common_Search.wsdl, you would edit the following line:

```
<soap:address location ="http://echo.oblix.com:5555/identity/oblix/apps/
userservcenter/bin/userservcenter.cgi"/>
```

4. For a Java client, you develop a proxy object and a Java client that submits the request.

   You use a WSDL-to-Java conversion tool to process the NetPoint WSDL file and automatically generate a Java proxy object for the IdentityXML request.

   An example of a WSDL-to-Java tool is IBM’s WSDL2Java tool that comes with the IBM WSDK.

If you are familar with .NET, you use the following process:

Task overview: Using WSDL to generate .NET IdentityXML requests

1. Look up the function that you wish to use in “IdentityXML Functions and Parameters” on page 67.
2. Edit the location information in the soap:address statement in a corresponding WSDL document.

   For example, for Common_Search.wsdl, you would edit the following line:

```
<soap:address location ="http://echo.oblix.com:5555/identity/oblix/apps/
userservcenter/bin/userservcenter.cgi"/>
```

3. Create .NET code that submits the request, as described in “.NET Implementation of WSDL” on page 52.

.NET Implementation of WSDL

In a .NET environment, you submit the correct WSDL common files to Visual Studio, which creates a.NET client. You edit the parameters in the generated client code, build the code, and use it as you would any other Web service.

Oblix provides samples for invoking Web services using C#.

Prerequisites for creating a .NET WSDL client:

1. Be sure your Web services directory is exposed through your Web server so that you can add Web references using Visual Studio.
2. Install .NET Studio 2003 with .NET Framework 1.1.
3. Install two Microsoft hot fixes:
   - The first fix to apply is for XML messaging with the .Net Framework, Hot Fix Rollup at http://support.microsoft.com?id=822411.
   - You also need to apply the fix for .NET Framework 1.1 WSDL with Visual Studio .NET 2003 Hot Fix Rollup at http://support.microsoft.com/?id=823639.

To generate a .NET WSDL client

1. Launch Visual Studio.
2. From the Visual C# Projects folder, select Console Application Template and click OK.
3. Click Project > Add Web Reference.
4. In the Add Web Reference dialog, select the location where you have the Web services directory.

   The Web services directory is the location of your Oblix WSDL input files.

   For example, this could be your local machine or your UDDI server.

   The WSDL files from the selected location are displayed.

5. Select the file containing the WSDL service that is relevant to your application.

   These are the files with names that contain the function that you want to work with and do not contain "_interface" in the file name. For example, for a search function, you would edit common_search.wsdl, not common_search_interface.wsdl.

   Example: common_search.wsdl.

   The service is displayed.

   Note: Be sure that the URL location in the WSDL file points to the URL of your installed COREid System.

   The displayed window will show an error, "No ports or methods were found on this page." You can ignore this error. What is important is shown in the right-hand pane on this page.

6. Click the Add Reference button in the pane on the right.

   A project window will appear that shows that the link to the Web reference has been added. Visual Studio creates the proxy object code, and puts all elements of the object in one file called reference.cs.

7. In the main window, add the proxy object code.

8. Specify the application, version, function parameters, and any other information required to complete the client code.

   A sample .NET client is provided in the directory oblix\WebServices\samples.

9. To compile the .NET proxy object code, click Build > Build solution.

   Once the solution is compiled, you should be able to run it like any other executable, from within Visual Studio or another location.

Java: Editing the WSDL File

Before generating the Java proxy object, you need to edit the implementation WSDL files. These are the files with names that contain the function that you want to work with and do not contain "_interface" in the file name. For example, for a search function, you would edit common_search.wsdl, not common_search_interface.wsdl.

Within the WSDL file, supply the correct host name, port number, and URL for the location of the appropriate cgi file. For example:

```
<soap:address location="http://echo.oblix.com:5555/identity/oblix/apps/
userservcenter/bin/userservcenter.cgi"/>
```

Generating the Java Proxy Object

You process the WSDL file using a toolkit that converts this file to Java. There are several of these toolkits available for no charge on the Web. The WSDL-to-Java toolkits enable you to produce Java code that contains most of the information that you need to build your HTTP/S client. The toolkit generates the SOAP envelope and IdentityXML payload.

For example, the IBM Web Services Toolkit (WSTK) is available at the following location:

http://www.alphaworks.ibm.com/aw.nsf/reqs/webservicestoolkit

Before installing the toolkit, follow the directions on this site to install the IBM WebSphere SDK for Web Services. From IBM Websphere SDK for Web Services version 5.0, open the help from the Start menu, go to Accessing a Webservice > from a standalone java application > Service Locator client.

This shows you how to run a sample client.

Additional steps for using the wsdl2java toolkit

1. If you have used the default installation, the application server and the toolkit create the folders wsdk_5 and wstk-3.3.

   You need all the jars in the path to use wsdl2java.

2. Set a few environment variables:

```
set PATH=C:\WSDK_v5\appserver\java\bin;%PATH%
set
CLASSPATH=.;c:\wstk-3.3;c:\wstk-3.3\lib\wstk.jar;c:\wstk-3.3\lib;c:\wstk-3.3\ser
vices\demos\common\client;c:\wstk-3.3\lib\wsdl4j.jar;c:\wstk-3.3\wsil4j\lib\wsil
4j.jar;c:\wstk-3.3\lib\xercesImpl.jar;c:\wstk-3.3\lib\xml-apis.jar;c:\wstk-3.3\l
ib\xalan.jar;c:\wstk-3.3\lib\bsf.jar;c:\wstk-3.3\lib\js.jar;c:\wstk-3.3\uddi4j\l
ib\uddi4j.jar;c:\wstk-3.3\uddi4j-wsdl\lib\uddi4j-wsdl.jar;c:\wstk-3.3\lib\log.ja
r;c:\wstk-3.3\lib\ibmjsse.jar;c:\wstk-3.3\lib\webservices-security.jar;c:\wstk-3
.3\lib\xss4j.jar;c:\wstk-3.3\lib\jaas.jar;c:\wstk-3.3\lib\certpath.jar;c:\wstk-3
.3\lib\activation.jar;c:\wstk-3.3\lib\mailapi.jar;c:\wstk-3.3\lib\IBMJCEProvider
.jar;c:\wstk-3.3\lib\IBMJCEfw.jar;c:\wstk-3.3\lib\US_export_policy.jar;c:\wstk-3
.3\lib\local_policy.jar;c:\wstk-3.3\axis;c:\wstk-3.3\axis\lib\axis.jar;c:\wstk-3
.3\axis\lib\jaxrpc.jar;c:\wstk-3.3\axis\lib\saaj.jar;c:\wstk-3.3\axis\lib\log4j-
1.2.4.jar;c:\wstk-3.3\axis\lib\commons-logging.jar;c:\wstk-3.3\axis\lib\commons-
discovery.jar;c:\wstk-3.3\Cloudscape\db2j.jar;;c:\wstk-3.3\Cloudscape\db2jtools.
jar;
set JAVA_HOME=C:\WSDK_v5\appserver\java
set LIB=C:\WSDK_v5\appserver\lib
set
CLASSPATH=%LIB%\axis.jar;%LIB%\jaxrpc.jar;%LIB%\commons-logging-api.jar;%LIB%\co
mmons-discovery.jar;%LIB%\j2ee.jar;%LIB%\qname.jar;%LIB%\xerces.jar;%LIB%\saaj.j
ar;%LIB%\webservices.jar;%CLASSPATH%;C:\wstk-3.3\bin\com\oblix\www\WSClient.jar
```

3. Generate the proxy object for the WSDL file.

   For example, the following command:

```
WSDL2Java -verbose
d:\oblix\sources\panacea\WebServices\WSDL\gm_viewgroupmembers.wsdl -o
c:\test
```

   creates all the required classes for login, request, and so on. The -o parameter provides a destination folder for the command output. The proxy object classes provide all of the components that you need to create a NetPoint request. Working with this proxy object is described in the following paragraphs.

Working With the Java Proxy Object

The proxy object that you create with the WSDL-to-Java toolkit contains files that correspond to everything that was in the original WSDL file. If you browse the output folder, several files are of particular interest. The following examples assume that you generated a proxy object for the search function:

oblixIDXML_Common_Search_BindingStub.java—This file creates a method that contains the authentication and request parameters.

OblixIDXML_Common_Search_ServiceLocator.xml—This file creates the SOAP address.

Authentication.java—This file creates a method for defining a login, password, and domain.

Request.java—This file creates a method for allowing you to set the value for application to be userservcenter, groupservcenter, or objservcenter. It also creates methods for reading the value of the IdentityXML function name, mode and version.

Params.java—This file provides methods for reading one or more variables, for example, noOfRecords or noOfFields. These correspond to the parameters that you can specify for the IdentityXML function.

To call the proxy object

1. Write a Java client to call these objects.
2. Compile the generated files and the java client and then run the client.
3. Be sure you receive the correct result.

   The tool TCPMon that comes with the IBM toolkit shows the XML/SOAP request that is sent to NetPoint and the SOAP response that NetPoint returns.

Note: Other vendors create WSDL-to-Java toolkits similar to the one described above. Note that if the generated code gives compiler errors, manual adjustments may be necessary.

Sample Generated Java Files

The following Java files illustrate BindingStub, ServiceLocator, Authentication, Request, and Params.java. In these examples, these files were generated for the search function using a WSDL-to-java tool. You do not edit these files. All of the required content is generated automatically. It is useful to know the contents of these files when you write your Java clients.

The BindingStub contains the authentication and request parameters. These parameters are shown in **bold**. The Java files that define the authentication and request methods are shown in "Sample Request.java File" on page 59 and "Sample Authentication.java file" on page 58.

Listing 16 Excerpt from Common_Search_BindingStub.java

```
/**
. . .
 * OblixIDXML_Common_Search_BindingStub.java
 *
 * This file was auto-generated from WSDL
 * by the Apache Axis WSDL2Java emitter.
 */
package com.oblix.www;
public class OblixIDXML_Common_Search_BindingStub extends
org.apache.axis.client.Stub implements com.oblix.www.OblixIDXMLPortType {
    private java.util.Vector cachedSerClasses = new java.util.Vector();
. . .
public OblixIDXML_Common_Search_BindingStub(javax.xml.rpc.Service service)
throws org.apache.axis.AxisFault {
        if (service == null) {
            super.service = new org.apache.axis.client.Service();
        } else {
            super.service = service;
. . .
">ObDisplay>obcanRequest");
            cachedSerQNames.add(qName);
            cls = com.oblix.www.ObcanRequest.class;
            cachedSerClasses.add(cls);
. . .
    public com.oblix.www.Oblix
oblixIDXML_Common_Search(com.oblix.www.Authentication authentication,
com.oblix.www.Request request) throws java.rmi.RemoteException {
        if (super.cachedEndpoint == null) {
            throw new org.apache.axis.NoEndPointException();
        }
```

```
      _call.addParameter(new javax.xml.namespace.QName("http://www.oblix.com/",
"request"), new javax.xml.namespace.QName("http://www.oblix.com/", ">request"),
com.oblix.www.Request.class, javax.xml.rpc.ParameterMode.IN);
. . .
```

The ServiceLocator file provides the SOAP address for the Search function. The address is shown in **bold**. The address is taken from the Common_Search.wsdl file when you generate the proxy object.

Listing 17 Common_Search_ServiceLocator.java

```
/**
* OblixIDXML_Common_Search_ServiceLocator.java. This file was auto-generated
* from WSDL by the Apache Axis WSDL2Java emitter.
*/
package com.oblix.www;
public class OblixIDXML_Common_Search_ServiceLocator extends
org.apache.axis.client.Service implements
com.oblix.www.OblixIDXML_Common_Search_Service {
// Use to get a proxy class for OblixIDXML_common_search_Port
private final java.lang.String OblixIDXML_common_search_Port_address = "http://
njadhavxp.oblix.com:1234/identity/oblix/apps/userservcenter/bin/
userservcenter.cgi";
public java.lang.String getOblixIDXML_common_search_PortAddress() {
return OblixIDXML_common_search_Port_address;
// The WSDD service name defaults to the port name.
private java.lang.String OblixIDXML_common_search_PortWSDDServiceName =
"OblixIDXML_common_search_Port";
public java.lang.String getOblixIDXML_common_search_PortWSDDServiceName() {
return OblixIDXML_common_search_PortWSDDServiceName;
}
public void setOblixIDXML_common_search_PortWSDDServiceName(java.lang.String
name) {
OblixIDXML_common_search_PortWSDDServiceName = name;
}
public com.oblix.www.OblixIDXMLPortType getOblixIDXML_common_search_Port()
throws javax.xml.rpc.ServiceException {
java.net.URL endpoint;
. . .
public com.oblix.www.OblixIDXMLPortType
getOblixIDXML_common_search_Port(java.net.URL portAddress) throws
javax.xml.rpc.ServiceException {
try {
com.oblix.www.OblixIDXML_Common_Search_BindingStub _stub = new
com.oblix.www.OblixIDXML_Common_Search_BindingStub(portAddress, this);
_stub.setPortName(getOblixIDXML_common_search_PortWSDDServiceName());
return _stub;
. . .
/**
* For the given interface, get the stub implementation. If this service has no
* port for the given interface, then ServiceException is thrown.
. . .
```

Authentication.java defines the methods for setting the login id, password, and domain. You pass the actual values for these methods in the Java client. See “Sample Java Client” on page 62 for details.

Listing 18 Sample Authentication.java file

```
/**
* Authentication.java
*
* This file was auto-generated from WSDL
* by the Apache Axis WSDL2Java emitter.
*/
package com.oblix.www;
public class Authentication  implements java.io.Serializable {
private java.lang.String login;
private java.lang.String password;
private java.lang.String domain;
public Authentication() { }
public java.lang.String getLogin() {
return login;
}
public void setLogin(java.lang.String login) {
this.login = login;
}
public java.lang.String getPassword() {
return password;
}
public void setPassword(java.lang.String password) {
this.password = password;
}
public java.lang.String getDomain() {
return domain;
}
public void setDomain(java.lang.String domain) {
this.domain = domain;
}
private java.lang.Object __equalsCalc = null;
public synchronized boolean equals(java.lang.Object obj) {
if (!(obj instanceof Authentication)) return false;
Authentication other = (Authentication) obj;
if (obj == null) return false;
if (this == obj) return true;
if (__equalsCalc != null) {
return (__equalsCalc == obj);
}
__equalsCalc = obj;
boolean _equals;
_equals = true &&
((login==null && other.getLogin()==null) ||
(login!=null &&
login.equals(other.getLogin()))) &&
. . .
```

Request.java provides the method that allows you to define an application to be userservcenter, groupservcenter, or objservcenter. It also also provides the method that allows you to call the IdentityXML function and its parameters, the mode, and the version. You create a Java client to pass the actual values for these items. See “Sample Java Client” on page 62.

Listing 19 Sample Request.java File

```
/**
* Request.java
* This file was auto-generated from WSDL by the Apache Axis WSDL2Java emitter.
*/
package com.oblix.www;
public class Request  implements java.io.Serializable {
private com.oblix.www.Params params;
private java.lang.String version;  // attribute
private com.oblix.www.Application application;  // attribute
private java.lang.String function;  // attribute
private java.lang.String mode;  // attribute
public Request() {
}
public com.oblix.www.Params getParams() {
return params;
}
public void setParams(com.oblix.www.Params params) {
this.params = params;
}
public java.lang.String getVersion() {
return version;
}
public void setVersion(java.lang.String version) {
this.version = version;
}
public com.oblix.www.Application getApplication() {
return application;
}
public void setApplication(com.oblix.www.Application application) {
this.application = application;
}
public java.lang.String getFunction() {
return function;
}
public void setFunction(java.lang.String function) {
this.function = function;
}
public java.lang.String getMode() {
return mode;
}
public void setMode(java.lang.String mode) {
this.mode = mode;
. . .
```

Params.java defines the methods for reading and setting the IdentityXML parameters, for example, noOfRecords or noOfFields. You set the values for these variables in the Java client, as illustrated in “Sample Java Client” on page 62.

Listing 20 Params.java File Generated for Common_Search.wsdl

```
/**
 * Params.java
 *
 * This file was auto-generated from WSDL
 * by the Apache Axis WSDL2Java emitter.
 */
package com.oblix.www;
public class Params  implements java.io.Serializable {
    private java.lang.String tab_Id;
    private java.math.BigInteger startFrom;
    private java.math.BigInteger noOfRecords;
    private java.math.BigInteger noOfFields;
    private com.oblix.www.BooleanValuesType showAllResults;
    private java.lang.String sortBy;
    private com.oblix.www.SortOrderValuesType sortOrder;
    private java.lang.String[] attrName;
    private com.oblix.www.SearchParams searchParams;
     public Params() {
    }
    public java.lang.String getTab_Id() {
        return tab_Id;
    }
    public void setTab_Id(java.lang.String tab_Id) {
        this.tab_Id = tab_Id;
    }
    public java.math.BigInteger getStartFrom() {
        return startFrom;
    }
    public void setStartFrom(java.math.BigInteger startFrom) {
        this.startFrom = startFrom;
    }
    public java.math.BigInteger getNoOfRecords() {
        return noOfRecords;
    }
    public void setNoOfRecords(java.math.BigInteger noOfRecords) {
        this.noOfRecords = noOfRecords;
    }
    public java.math.BigInteger getNoOfFields() {
        return noOfFields;
    }
    public void setNoOfFields(java.math.BigInteger noOfFields) {
        this.noOfFields = noOfFields;
. . .
```

The following is a sample Oblix.java file generated from Common_Search.wsdl. This defines the methods for getting the search results and creating the output for the search request.

Listing 21 Sample Oblix.java file Generated from Common_Search.wsdl

```
/**
 * Oblix.java
 *
 * This file was auto-generated from WSDL
 * by the Apache Axis WSDL2Java emitter.
 */

package com.oblix.www;

public class Oblix  implements java.io.Serializable {
    private com.oblix.www.ObSearch obSearch;
    private com.oblix.www.ObError obError;
    private com.oblix.www.ObNavbar obNavbar;
    private com.oblix.www.ObSearchForm obSearchForm;
    private java.lang.String obStatus;

    public Oblix() {
    }

    public com.oblix.www.ObSearch getObSearch() {
        return obSearch;
    }

    public void setObSearch(com.oblix.www.ObSearch obSearch) {
        this.obSearch = obSearch;
    }

. . .
    }

    public com.oblix.www.ObSearchForm getObSearchForm() {
        return obSearchForm;
    }

    public void setObSearchForm(com.oblix.www.ObSearchForm obSearchForm) {
        this.obSearchForm = obSearchForm;
  . . .
```

Sample Java Client

The following is a sample Java client for the search function. This client generates a search in the Group Manager. There are two conditions in this client, one to search for a cn value of “basic” and one to search for a description value of “Customer.” You create clients like this one to make use of the Java proxy object.

Important: As shown in the example below, you must set the version string to NPWSDL1.0. For example, myrequest.setVersion("NPWSDL1.0");.

Listing 22 Sample Java Client for Common_Search.wsdl

```
// Search for groups contains where cn contains "Basic" and description contains
"Customer"
import java.math.*;
import com.oblix.www.*;

public class cuser10k1_asort_member{
public static void main( String[] args) throws Throwable
{

OblixIDXML_Common_Search_ServiceLocator loc = new
OblixIDXML_Common_Search_ServiceLocator();
OblixIDXMLPortType port = loc.getOblixIDXML_common_search_Port();
Authentication myauth =  new Authentication();
myauth.setLogin("cuser10k1");
myauth.setPassword("oblix");
Request myrequest =  new Request();
myrequest.setFunction("search");
myrequest.setApplication("groupservcenter");
myrequest.setVersion("NPWSDL1.0");
Params myparams =  new Params();
myparams.setSortBy("uniqueMember");
myparams.setNoOfFields(new BigInteger("1"));
SearchParams mysearchParams = new SearchParams();
com.oblix.www.Condition[]  myconditionarray =  new com.oblix.www.Condition[2];
mysearchParams.setCondition(myconditionarray);
Condition mycondition_0 =  new Condition();
mycondition_0.setSearchAttr("cn");
mycondition_0.setSearchOperation(new SearchOperAttrType("OSM"));
mycondition_0.setSearchString("Basic");
myconditionarray[0] = mycondition_0;
Condition mycondition_1 =  new Condition();
mycondition_1.setSearchAttr("description");
mycondition_1.setSearchOperation(new SearchOperAttrType("OSM"));
mycondition_1.setSearchString("Customer");
myconditionarray[1] = mycondition_1;
myparams.setSearchParams(mysearchParams);
myrequest.setParams(myparams);
Oblix output = port.oblixIDXML_Common_Search(myauth, myrequest);
System.out.println("output.getObStatus"());
```

```
}
}
```

Explanation of the Sample Java Client

The example above shows a Java client that uses the IdentityXML Search function. This client takes its input from the Java proxy object that is created when you translate common_search.wsdl file into Java using the WSDL-to-Java tool. Parameter values supplied in the Java client are specific to the NetPoint installation. To obtain these parameter values, you need to know how to obtain these values by using NetPoint, or you need to coordinate with a NetPoint administrator to obtain these values. The parameter names are standard IdentityXML parameter names, and they are defined in the params.java file in the Java proxy object.

Other specifics of this client are as follows:

OblixIDML_Common_Search_ServiceLocator—This calls the ServiceLocator function defined in the Common_Search_ServiceLocator.java file in the Java proxy object. This function reads in the SOAP address for this service.

OblixIDXMLPortType—This calls the PortType function defined in the Common_Search_ServiceLocator.java file in the proxy object. The PortType function reads in the URL for the search application.

Authentication—This is a function to define the login.

setLogin—This passes a value to the login function defined in Authentication.java.

setPassword—This passes a value to the password function defined in Authentication.java.

setFunction—This passes the name of the Identity XML function to the Function parameter defined in Request.java.

setApplication—This passes the name of the COREid application in which the IdentityXML function will be used to the application function defined in Request.java.

setVersion—To use WSDL, the version *must* be set to NPWSDL1.0. For example: myrequest.setVersion("NPWSDL1.0");

setSortBy—This passes a value to the SortBy function defined in Params.java. This corresponds to the SortBy parameter that is used with the IdentityXML search function.

setNoOfFields—This passes a value to the NoOfFields function defined in Params.java. This corresponds to the NoOfFields parameter that is used with the IdentityXML search function.

setCondition—This passes a Condition delimiter to the equivalent function defined in Params.java. This corresponds to the Condition delimiting tag for the IdentityXML search parameters.

setSearchAttr—This passes a value to the SearchAttr function defined in Params.java. This corresponds to the SearchAttr parameter that is used with the IdentityXML search function.

setSearchOperation—This passes a value to the SearchOperation function defined in Params.java. This corresponds to the IdentityXML parameter SearchOperation.

setSearchString—This passes a value to the SearchString function defined in Params.java. This corresponds to the IdentityXML parameter SearchString.

Sample SOAP Request That References WSDL

Note that a SOAP request that makes use of WSDL must have the string version="NPWSDL1.0" in the request. An example:

Listing 23 Sample WSDL SOAP request

```
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas-xmlsoap.org/soap/envelope/"
xmlns:oblix="http://www.oblix.com">
<SOAP-ENV:Body>
<oblix:request application="userservcenter" function="view"
mode="dataonly" xmlns="http://www.oblix.com/" version="NPWSDL1.0">
<oblix:attributeParams>
<oblix:attrName>sn</oblix:attrName>
</oblix:attributeParams>
</oblix:request>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Making WSDL Functions Available Using UDDI

The Universal Description, Discovery, and Integration (UDDI) registry is a database for people who require WSDL functions. UDDI provides a way to publish and categorize Web services created using WSDL. UDDI is analogous to the White Pages or Yellow pages, in that you can browse the UDDI registry for functions that you need, and you can add new functions to the registry. Global UDDI registries that can be accessed by anyone from any organization are provided by companies such as IBM and Microsoft. Instructions for creating and using a registry account is provided at these UDDI sites. Other organizations have their own internal UDDI registries.

As an illustration of how people make use of UDDI, suppose a car dealer needs to interact with remote dealers. This dealer can use their organization's UDDI registry as a type of Yellow Pages where they can find Web-based services for locating other dealers. To continue the illustration, suppose the UDDI registry contains the directory software_publishers/identity management/Oblix. The hypothetical car dealer might retrieve an entry in this directory for a Web service that allows users to find remote dealers. The entry would consist of a URL that points to a WSDL file that is capable of generating the desired search request.

In general, UDDI registries contain the following information for each Web service:

- The business name, for instance, Oblix
- The service (sometimes called an interface in UDDI parlance), which is the XML function, for example, view, plus the input and output parameters in XSD format.
- The implementation, which is the URL that points to the corresponding WSDL.

Follow the conventions used in your organization for locating the appropriate UDDI registries.

When you work with NetPoint's Web services functionality, you can register your own functions in UDDI. If you want to build an interface to interact with the IdentityXML system, you can use UDDI to find the appropriate WSDL definitions and use these definitions to develop the Java client that interacts with the IdentityXML service.

Sample UDDI registration programs in .NET and Java format are provided in the following location:

webpass_install_dir\oblix\WebServices\UDDI\dotnet

and

webpass_install_dir\oblix\WebServices\UDDI\java

There are readme files in both directories.These directories also contain a sample file for testing the function after it is registered.

3 IdentityXML Functions and Parameters

IdentityXML functions allow you to programmatically perform operations in the COREid applications. For example, using IdentityXML, you can perform functions such as:

- Finding users, adding users, changing user attributes in a COREid application.
- Creating a group, subscribing a user to a group.
- Determining if a user has the right to perform an operation.

This chapter describes each IdentityXML function and the parameters for each function.

This chapter contains the following sections:

Note: For an overview of IdentityXML, see "IdentityXML and NetPoint Web Services" on page 19.

About IdentityXML Usage

IdentityXML allows you to write programs to perform various actions in the COREid applications. The COREid applications are the following:

User Manager—The User Manager enables administrators to add, modify, and delete user identities. The User Manager typically enables end users to view other users and to modify their own identity information. The users that a person can view and the identity information that someone can modify depends on the privileges granted by a NetPoint administrator.

Group Manager—If you are an administrator, the Group Manager enables you to create or delete groups, and enables users to view groups and to subscribe or unsubscribe from groups. A user's ability to create and delete groups and to subscribe to various groups depends on the privileges granted by a NetPoint administrator.

Organization Manager—If you are an administrator, the Organization Manager enables you to create and delete organizations and other objects (such as floor plans and assets) that do not belong in the User Manager or Group Manager. A user's ability to view objects, add them, and modify them depends on the privileges granted by a NetPoint administrator.

You can create programs with IdentityXML that perform actions such as adding a user to the User Manager or changing the attribute values of an entry in the Organization Manager. You can also create programs that make use of the identity workflow capability of the COREid System. A COREid workflow enables you to link actions into an automated chain of events that are presented in the COREid System as a series of steps. When you create a workflow definition, you specify who is to perform each action, possibly calling out to external applications at one or more points in the process.

Overview of IdentityXML Functions and Parameters

The following sections describe IDXML functions and parameters. Parameters that are labeled *required* must be used in the function statement. *Optional* parameter names and values may be omitted, in which case a default may apply. Entering an optional parameter's name but not its value is an error. In the following sections, parameter *defaults* are provided. If a default value has been omitted from the description, there is no default for the parameter.

The values for many parameters are the exact DN values as they appear in the directory rather than the display values. To find the DN values, you can use a tool that allows you to browse the directory and display DN entries. An example of such a tool is ldp.exe provided with Windows 2000 systems.

Other attributes are the LDAP schema names of the attribute, rather than the display name.

To find schema names for an attribute

1. Taking the User Manager as an example, click COREid System Configuration > User Manager > User Manager Configuration > Configure Tab.
2. Click on the link for the tab.
3. Click Modify Attributes.

   An applet appears.The top left corner shows a list of schema names for the attribute, and the top right corner shows the display names of the attributes, where you can locate the attribute you want to refer to. The attribute names for that tab are displayed in the field identified as Attribute.

Function Types

IdentityXML functions are of four basic types:

- Common functions—these are functions that are applicable to every COREid application.
- User Manager functions
- Group Manager functions
- Organization Manager functions

Each function performs one of two basic activities:

- Testing to see if a particular person has the right to perform a specific operation
- Actually performing the operation (for instance, finding a user)

Common Functions

The following are functions used throughout the COREid applications. Note that all functions follow a similar syntax:

<oblix:request application="userservcenter|groupservcenter|objservcenter" function="*function name*">

For example:

<oblix:request application="userservcenter" function="search">

Search for entries based on some criteria

Function: search

Request example:
```
<oblix:request application="userservcenter"
function="search">
```

Description: Search for an entry or entries. The entries must be in a searchbase accessible to the user.

Works with: Group, Organization, and User Manager.

Results: The output is defined by the schema file oblix\WebServices\XMLSchema\component_search.xsd

Output schema: oblix\WebServices\XMLSchema\searchResults.xsd

WSDL file: *WebPass_install_dir*\oblix\WebServices\WSDL\common_search.wsdl

Parameters:

`SLkn`	Required (old syntax). See "Pre-NetPoint 6.5 Search Parameters" on page 140.
`SStn`	Required (old syntax). See "Pre-NetPoint 6.5 Search Parameters" on page 140.
`STyn`	Required (old syntax). See "Pre-NetPoint 6.5 Search Parameters" on page 140.
`SearchAttr`	Required (new syntax). See "NetPoint 6.5 Search Parameters" on page 142.
`SearchOperation`	Required (new syntax). See "Pre-NetPoint 6.5 Search Parameters" on page 140.
`SearchString`	Required (new syntax). See "Pre-NetPoint 6.5 Search Parameters" on page 140.
`attrname`	Optional. If no value is given, the default table view attributes are used. See "Pre-NetPoint 6.5 Attribute Parameters" on page 145 or "NetPoint 6.5 and Higher Attribute Parameters" on page 149.

`noOfFields`	Optional. See “Pre-NetPoint 6.5 Attribute Parameters” on page 145 or “NetPoint 6.5 and Higher Attribute Parameters” on page 149.
`noOfRecords`	The maximum number of entries to return in the search results. Overridden by the showAllResults parameter. Rules: Optional. Single value, an integer value >=1. Default: A value obtained from the defaultDisplayResultVal parameter in the oblixbaseparams.xml catalog. Otherwise, this value is obtained from the custom cookie.
`showAllResults`	Specifies that all results of the search be returned. If the value is true, it overrides the value of the noOfRecords parameter. Rules: Optional. True or false. Default: false, meaning return results up to the limit imposed by the noOfRecords parameter.
`sortBy`	What attribute to use to sort the results. Rules: Optional. Single value. Default: if no value is specified, the class attribute for the structural object class of the tab specified by tab_id is used.
`sortOrder`	The sort order, ascending or descending. Rules: Optional. Single value, ascending or descending Default: ascending
`startFrom`	Use this parameter for a long list of search results, to skip over a selected number of items and start the list with a specified item. For example, if 100 entries were found by the search, entering a value of 80 for this parameter gives a response showing only items 80 through 100. Rules: Optional. Single value, integer. Default: 1, to start displaying from the beginning of the search results list.

`tab_id`	The name of the tab that describes the information category you want to search within. For User Manager and Group Manager only one tab is allowed. For Organization Manager, multiple tabs are allowed. If omitted, NetPoint uses a default value for tab_id of the leftmost tab. Oblix recommends that you always provide a value for tab_id. Organization Manager allows you to change the order in which tabs are displayed. If you rely on the default tab_id, your portal functions would be affected. The tab_id is a number. To get the number, go to the configuration menu for the application. Choose configure tab. Position the cursor on the tab whose tab_id you want, and right click, then click the tab name whose tab_id you want. Select Open in new window. In the URL displayed at the top of the page, you find the value for tab_id. Rules: Optional. Single value Default:For User Manager and Group Manager, which have only a single tab, that tab is assumed. For Organization Manager, which has multiple tabs, the leftmost tab is assumed.

Search Example

To search in the User Manager for all entries that have "john" in the name:

```
<?xml version="1.0"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas-xmlsoap.org/soap/
envelope/">
<SOAP-ENV:Body>
   <oblix:authentication xmlns:oblix="http://www.oblix.com" type="basic">
      <oblix:login>J.Smith</oblix:login>
      <oblix:password>J.Smith</oblix:password>
   </oblix:authentication>
   <oblix:request application="userservcenter" function="search">
      <oblix:SearchParams>
         <oblix:Condition>
              <oblix:SearchAttr>cn</oblix:SearchAttr>
              <oblix:SearchOperation>OSM</oblix:SearchOperation>
              <oblix:SearchString>john</oblix:SearchString>
         </oblix:Condition>
      </oblix:SearchParams>
   </oblix:request>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Search for all pending, completed, or all tickets

Function: workflowTicketSearch

Request example:
```
<oblix:request application="userservcenter"
function="workflowTicketSearch">
```

Description: Search for pending, or completed, or all workflow requests.

Works with: Group Manager, Organization Manager, User Manager

Results: The output is defined by the schema file oblix\WebServices\XMLSchema\component_search.xsd.

Output schema: oblix\WebServices\XMLSchemasearch\Results.xsd

WSDL file: *WebPass_install_dir*\oblix\WebServices\WSDL\common_workflowTicketSearch.wsdl

Parameters:

`requestType` The request queue type to search.

`incomingRequests`—Requests you need to process.

`outgoingRequests`—Requests you have originated.

Rules: Required. Single value.

`targetApplication`
The application to search for tickets. To search all applications, use the value allApplications. To search a specific application, enter the application name:

`groupservcenter`—For Group Manager

`objservcenter`—For Organization Manager

`userservcenter`—For User Manager

Rules: Required. Single value.

`ticketType` The status type for the requests to be searched. There are three possible entries:

	`WfAllTickets`—Search for all requests, regardless of status. `WfCompletedTickets`—Search for requests that have been completely processed. `WfPendingTickets`—Search for requests that are pending, waiting to be processed. Rules: Required. Single value.
`days`	Look for requests issued in the past *n* days. NetPoint considers a day to be the 24-hour period from when the ticket was created, not a calendar day. Rules: Optional. Single value, an integer >=1. Default: 0, meaning look as far back as the oldest request.
`noOfRecords`	A maximum number of entries to be returned in the search results.This is overridden by the showAllResults parameter. Rules: Optional. Single value, an integer value >=1. Default: A value obtained from the defaultDisplayResultVal parameter in the oblixbaseparams.xml catalog. Otherwise this value is obtained from the custom cookie.
`sortBy`	What attribute to use to sort the results. Rules: Optional. Single value. Default: if no value is specified, the class attribute of the structural object class of the tab specified by tab_id is used. For workflow tickets, the class sorting attribute can have only one of the following values: `obticketid`—For Ticket Number `obapp`—For Application Name `obactionname`—For Action `obwfstatus`—For Status `obwftypename`—For Request Type `obtargetdn`—For Requested For

`obcurrentdn`—For Requested by
`obactordn`—For Action Taker
`obdateprocessed`—For Date Processed
`oblockedby`—For Locked By
`obsubflow`—For Subflow Number

If the attribute is invalid, an error is returned, such as "Invalid value for parameter sortBy." If no attribute is specified, the default is the first attribute (most likley obticketid) in the admininstrator-configured workflow ticket search table. You can see this table by looking at the COREid System Console > Common Configuration > Configure Workflow Panels > Ticket Search Table.

`sortOrder` — The sort order, ascending or descending. An invalid order gives an error message.

Rules: Optional. Single value, ascending or descending.

Default: ascending

`startFrom` — Use this parameter for a long list of search results, to skip a number of items and start the list with a specified item. For example, if 100 entries were found by the search, entering a value of 80 for this parameter gives a response showing only items 80 through 100.

Rules: Optional. Single value, integer.

Default: 1, to start from the beginning of the search results list.

workflowTicketSearch Notes

If the mode is dataonly, the possible values for obwfstatus are integers, as follows:

```
Unknown = -1
Success = 0
Failed = 1
PendingUser = 2
PendingSubflow = 3
PendingPreAction = 4
PendingPostAction = 5
PendingUserInPre = 6
PendingUserInPost = 7
LastStepDone = 8
Asynch = 9
PendingExecution = 10
Cancelled = 11
PendingPreNotify = 12
PendingPreSubflow = 13
PendingPostNotify = 14
TriggerSubflows = 15
ForceCommit = 16
Retry = 17
PendingRetry = 18
```

For the output [integer/string], the “store-as” is an integer. The string is the value displayed in the u.i.

workflowTicketSearch Example

To search for all of your incoming tickets in the User Manager:

```
<?xml version="1.0"?>
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas-xmlsoap.org/soap/envelope/">
<SOAP-ENV:Body>
   <oblix:authentication xmlns:oblix="http://www.oblix.com" type="basic">
        <oblix:login>J.Smith</oblix:login>
        <oblix:password>J.Smith</oblix:password>
   </oblix:authentication>
<oblix:request application="userservcenter" function="workflowTicketSearch">
   <oblix:params>
        <oblix:tab_id>Employees</oblix:tab_id>
        <oblix:requestType>incomingRequests</oblix:requestType>
        <oblix:ticketType>allTickets</oblix:ticketType>
   </oblix:params>
</oblix:request>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Get information on a particular workflow ticket

Function: workflowTicketInfo

Request example:
```
<oblix:request application="userservcenter"
function="workflowTicketInfo">
```

Description: Get information about a specific request.

Works with: Group Manager, Organization Manager, User Manager.

Results: The output is defined by the schema file oblix\WebServices\XMLSchema\wfTicketInfo.xsd

Output schema: If the operation is successful, it returns the profile of the group, according to the following XML Schema: oblix\WebServices\XMLSchema\wfTicketInfo.xsd

WSDL file: *WebPass_install_dir*\oblix\WebServices\WSDL\common_workflowTicketInfo.wsdl

Parameters:

`workflowInstanceDn`

The DN of the workflow for which information is required. The DN for the workflow is shown in the workflow definition view. See the *NetPoint 7.0 Administration Guide Volume 1*.

Rules: Required. Single DN value.

`workflowStepInstanceId`

A step in the workflow specified by workflowInstanceDn for which information is required.

Rules: Required. Single integer value.

Resume asynchronous workflows

Function: asynchResumeWorkflowProcess

Request example:
```
<oblix:request application="asynch"
function="asynchResumeWorkflowProcess">
```

Description: This function allows for the continuation of a workflow in which an Identity Event API call returned a status of STATUS_PPP_WF_ASYNC. The asyncResumeWorkflowProcess function takes a workflow instance DN and a step ID as input.

asynch_retcode =0 to resume the workfkow
asynch_retcode =1 to abort the workflow
default value =0

See “Identity Event Plug-in API” on page 171.

One or more of the parameters described as optional, below, must be provided, depending on the requirements of the particular workflow.

Works with: Asynchronous workflows.

Output schema: Currently there is a bug which always produces the output in html format like this:

<html>The action completed successfully. Please refer to the workflow page.
</html>

WSDL file: *WebPass_install_dir*\oblix\WebServices\WSDL\common_asynchResumeWorkflowProcess.wsdl

Parameters:

`workflowInstanceDn`
The DN of the workflow for which information is required. The DN for the workflow is shown in the workflow definition view. See the *NetPoint 7.0 Administration Guide Volume 1*.

Rules: Required. Single DN value.

`workflowStepInstanceId`
A step in the workflow specified by workflowInstanceDn for which information is required.

Rules: Required. Single integer value.

`attrName` Optional. See “Pre-NetPoint 6.5 Attribute Parameters” on page 145 or “NetPoint 6.5 and Higher Attribute Parameters” on page 149.

`attrOldValue`	Optional. See “Pre-NetPoint 6.5 Attribute Parameters” on page 145 or “NetPoint 6.5 and Higher Attribute Parameters” on page 149.
`attrOperation`	Optional. See “Pre-NetPoint 6.5 Attribute Parameters” on page 145 or “NetPoint 6.5 and Higher Attribute Parameters” on page 149.
`attrValue`	Optional (old syntax). See “Pre-NetPoint 6.5 Attribute Parameters” on page 145.
`attrNewValue`	Optional (new syntax). See “NetPoint 6.5 and Higher Attribute Parameters” on page 149.
`NoOfFields`	Optional. See “Pre-NetPoint 6.5 Attribute Parameters” on page 145 or “NetPoint 6.5 and Higher Attribute Parameters” on page 149.

Example 1

```
<?xml version="1.0"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas-xmlsoap.org/soap/envelope/">
<SOAP-ENV:Body>
<oblix:authentication xmlns:oblix=http://www.oblix.com" type="basic">
<oblix:login>npuser</oblix:login>
<oblix:password>password</oblix:password>
</oblix:authentication>
<oblix:request function="asynchResumeWorkflowProcess">
<oblix:params>
<oblix:param name="workflowInstanceDn">obwfinstanceid=43598793ab7364cd2322ea,
obcontainerId=workflowInstances,o=Oblix,ou=Apps, o=wachovia</oblix:param>
<oblix:param name="workflowStepInstanceId">1</oblix:param>
<oblix:param name="asynch_retcode">0</oblix:param>
<oblix:param name="attrName_1">racfdefaultgroup</oblix:param>
<oblix:param name="attrValue_1">defaultGroupNameValue</ oblix:param>
<oblix:param name="attrName_2">anattribute2</oblix:param>
<oblix:param name="attrValue_2">avalue2</oblix:param>
<oblix:param name="attrValue_3">uid</oblix:param>
<oblix:param name="attrValue_3">targetUID</oblix:param>
<oblix:param name="attrOperation">ADD</oblix:param>
</oblix:params>
</oblix:request>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Example 2

```
<?xml version="1.0"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas-xmlsoap.org/soap/envelope/">
<SOAP-ENV:Body>
<oblix:authentication xmlns:oblix="http://www.oblix.com" type="basic">
<oblix:login>authenticationAttribute</oblix:login>
<oblix:password>authenticationPassword</oblix:password>
</oblix:authentication>
<oblix:request function="asynchResumeWorkflowProcess">
<oblix:params>
<oblix:param name="workflowInstanceDn">obwfinstanceid=wfinstanceid,
obcontainerId=workflowInstances,o=Oblix,ou=Apps, o=mycompany</oblix:param>
<oblix:param name="workflowStepInstanceId">1</oblix:param>
<!-- See Return codes at the bottom of this file -->
<oblix:param name="asynch_retcode">0</oblix:param>

<!-- Add the attributes required by the workflow in the order the -->
<!-- workflow expects them. Include even the optional and hidden fields. -->
<!-- Start with n=1 -->
<oblix:param name="attrName_n">attr. name</oblix:param>
<oblix:param name="attrValue_n">attr. value</oblix:param>
<!-- ... all other workflow expected attributes -->
<!-- The operation depends of what you want to do with the attributes. -->
<!-- In this case I know attribute does not currently exist in the user -->
<!-- entry so I want to add them, however, you might want to replace the -->
<!-- values of the attributes, etc. -->
<oblix:param name="attrOperation">ADD</oblix:param>
</oblix:params>
</oblix:request>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>

<!-- The values for the return code, async_retcode, is as follows: -->
<!-- 0 - Success -->
<!-- 1 - Action Failed -->
<!-- -11 - Pre-Action Failed -->
<!-- -12 - Post-Action Failed -->
<!-- -13 - External-Action Failed -->
```

Subscribe self to group

Function: subscribe

Request example: `<oblix:request application="groupservcenter" function="subscribe">`

Description: Add (subscribe) yourself to a group. The response returns the profile for the group.

Works with: Group Manager.

Results: The output is the profile of the group, defined by the schema file oblix\WebServices\XMLSchema\gsc_groupprofile.xsd.

Output schema: If operation is successful, it returns the profile of the group, according to the following XML Schema.

oblix\WebServices\XMLSchema\gsc_groupprofile.xsd

If operation fails, you will get an error message like:

```
<SOAP-ENV:Envelope>
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet href="0" type="text/xsl"?>
<Oblix>
<ObError>
<ObRequestInfo>
161660048
</ObRequestInfo>
<ObTextMessage>
You do not have access rights.
</ObTextMessage>
</ObError>
</Oblix>
</SOAP-ENV:Envelop>
```

WSDL file: *WebPass_install_dir*\oblix\WebServices\WSDL\gm_subscribe.wsdl

Parameters:

`uid` In this case, uid means the DN of the group being subscribed to.

Rules: Required. Single value, a DN.

Unsubscribe self from group

Function: unsubscribe

Request example: `<oblix:request application="groupservcenter" function="unsubscribe">`

Description: Remove (unsubscribe) yourself from a group.

Works with: Group Manager.

Results*:* The response returns the profile of the group, defined by the schema file: oblix\WebServices\XMLSchema\gsc_groupprofile.xsd.

Output schema: If operation is successful, it returns the profile of the group, according to the following XML Schema: oblix\WebServices\XMLSchema\gsc_groupprofile.xsd

If operation fails, you will get an error message like:

```
<SOAP-ENV:Envelope>
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet href="0" type="text/xsl"?>
<Oblix>
<ObError>
<ObRequestInfo>
161660048
</ObRequestInfo>
<ObTextMessage>
You do not have access rights.
</ObTextMessage>
</ObError>
</Oblix>
</SOAP-ENV:Envelop>
```

WSDL file: *WebPass_install_dir*\oblix\WebServices\WSDL\ gm_unsubscribe.wsdl

Parameters:

`uid` The DN of the group being unsubscribed from.

Rules: Required. Single value, a DN.

Subscribe user to group

Function: subscribeUserToGroup

Request example: `<oblix:request application="groupservcenter" function="subscribeUserToGroup">`

Description: Subscribe a user other than yourself to a group. The other user does not need to be logged in.

Works with: Group Manager.

Results*:* The output is the profile of the group, defined by the schema file oblix\WebServices\XMLSchema\gsc_groupprofile.xsd

WSDL file: *WebPass_install_dir*\oblix\WebServices\WSDL\gm_subscribeUserToGroup.wsdl

Parameters:

`uid` The DN of the group entry.

`proxysourceuid` The DN for a non-logged-in user (proxy user) who is being subscribed.

Rules: Required. Single value, a DN.

Example

To subscribe Robert Fulton to a group:

```
<?xml version="1.0"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas-xmlsoap.org/soap/
envelope/">
<SOAP-ENV:Body>
   <oblix:authentication xmlns:oblix="http://www.oblix.com" type="basic">
        <oblix:login>J.Smith</oblix:login>
        <oblix:password>J.Smith</oblix:password>
   </oblix:authentication>
   <oblix:request application="groupservcenter"
        function="subscribeUserToGroup">
      <oblix:params>
         <oblix:proxysourceuid>
            cn=Robert Fulton, ou=Corporate, o=Company, c=US
         </oblix:proxysourceuid>
         <oblix:uid>
            cn=Marketing Team, ou=Marketing, o=Company, c=US
         </oblix:uid>
      </oblix:params>
   </oblix:request>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Unsubscribe user from group

Function: unsubscribeUserFromGroup

Request example: `<oblix:request application="groupservcenter" function="unsubscribeUserFromGroup">`

Description:	Unsubscribe a user other than yourself from a group. The other user does not need to be logged in.
Works with:	Group Manager.
Results*:*	The response returns the profile of the group, defined by the schema file: oblix\WebServices\XMLSchema\gsc_groupprofile.xsd.

Output schema:	If operation is successful, it returns the profile of the group, according to the following XML Schema.
	
	oblix\WebServices\XMLSchema\gsc_groupprofile.xsd
	
	If operation fails, you will get an error message like:
	
	`<SOAP-ENV:Envelope><?xml version="1.0" encoding="ISO-8859-1"?>`
	`<?xml-stylesheet href="0" type="text/xsl"?>`
	`<Oblix>`
	`<ObError>`
	`<ObRequestInfo>`
	`161660048`
	`</ObRequestInfo>`
	`<ObTextMessage>`
	`You do not have access rights.`
	`</ObTextMessage>`
	`</ObError>`
	`</Oblix>`
	`</SOAP-ENV:Envelop>`

WSDL file:	*WebPass_install_dir*\oblix\WebServices\WSDL\ gm_unsubscribeUserFromGroup.wsdl
Parameters:	
`uid`	The DN of the group being unsubscribed from. Rules: Required. Single value, a DN.
`proxysourceuid`	The DN for a non-logged-in user (proxy user) who is being unsubscribed. Rules: Required. Single value, a DN.

Example

To unsubscribe Robert Fulton from a group:

```
<?xml version="1.0"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas-xmlsoap.org/soap/envelope/">
<SOAP-ENV:Body>
   <oblix:authentication xmlns:oblix="http://www.oblix.com" type="basic">
      <oblix:login>J.Smith</oblix:login>
      <oblix:password>J.Smith</oblix:password>
   </oblix:authentication>
<oblix:request application="groupservcenter"
      function="unsubscribeUserFromGroup">
   <oblix:params>
      <oblix:proxysourceuid=cn=Robert Fulton, ou=Corporate, o=Company,
         c=US</oblix:proxysourceuid>
      <oblix:uid>cn=Marketing Team, ou=Marketing, o=Company,c=US</oblix:uid>
   </oblix:params>
   </oblix:request>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

User Manager Functions

If you are an administrator, the User Manager enables you to add, modify, and delete user identities. The User Manager typically enables end users to view other users and to modify their own identity information. The users that a person can view and the identity information that someone can modify depends on the privileges granted by a NetPoint administrator.

The following IdentityXML functions allow you to programmatically access the User Manager application. Note that all functions follow a similar syntax:

<oblix:request application="userservcenter" function="*name*">

For example:

<oblix:request application="userservcenter" function="canIViewUserProfile">

Functions to Test for Attribute Permissions

The following functions provide a yes or no response as to whether you or another user has read, write, delegate, and notify permissions set for a particular attribute.

Can I view a user's profile

Function: canIViewUserProfile

Request example:
```
<oblix:request application="userservcenter"
function="canIViewUserProfile">
```

Description: Verifies that you can view a user's profile.

WSDL file: *WebPass_install_dir*\oblix\WebServices\WSDL\um_CanIViewUserProfile.wsdl

Parameters:

`uid` The DN of the user whose profile you want to view.

Rules: Required. Single value, a DN.

Can I view an attribute in a user's profile

Function: canIViewUserProfileAttr

Request example:
```
<oblix:request application="userservcenter"
function="canIViewUserProfileAttr">
```

Description: Verifies that you can view a particular attribute in a user's profile.

WSDL file: *WebPass_install_dir*\oblix\WebServices\WSDL\um_CanIViewUserProfileAttr.wsdl

Parameters:

`uid` The DN of the user whose attribute you want to view.

Rules: Required. Single value, a DN.

`targetAttribute` The schema name (not the display name) for the desired attribute.

Rules: Required. Single value, a string.

Can I modify an attribute in a user's profile

Function: canIModifyUserProfileAttr

Request example:
```
<oblix:request application="userservcenter"
function="canIModifyUserProfileAttr">
```

Description: Verifies that you can change a particular attribute in a user's profile.

WSDL file: *WebPass_install_dir*\oblix\WebServices\WSDL\um_canIModifyUserProfileAttr.wsdl

Parameters:

`uid` The DN of the user whose attribute you want to change.

Rules: Required. Single value, a DN.

`targetAttribute` The schema name (not the display name) for the desired attribute.

Rules: Required. Single value, a string.

Can I modify an attribute in a user's profile using a workflow

Function: canIRequestUserAttrModification

Request example:
```
<oblix:request application="userservcenter"
function="canIRequestUserAttrModification">
```

Description: Verifies that you can change a particular attribute in a user's profile, using a workflow.

WSDL file: *WebPass_install_dir*\oblix\WebServices\WSDL\um_CanIRequestUserAttrModification.wsdl

Parameters:

`uid` The DN of the user whose attribute you want to change.

Rules: Required. Single value, a DN.

`targetAttribute` The schema name (not the display name) for the desired attribute.

Rules: Required. Single value, a string.

Can I create a new user

Function: canICreateUser

Request example:
```
<oblix:request application="userservcenter"
function="canICreateUser">
```

Description: Verifies that you can create a new user.

WSDL file: *WebPass_install_dir*\oblix\WebServices\WSDL\um_CanICreateUser.wsdl

Parameters:

`ObDomainName` A subtree within which a test is being requested.

Rules: Optional. Single value, a DN.

Default: if no value is provided, NetPoint checks to see if you have the tested rights in *any* domain.

Can I delete an existing user

Function: canIDeleteUser

Request example:
```
<oblix:request application="userservcenter"
function="canIDeleteUser">
```

Description: Verifies that you can delete an existing user.

WSDL file: *WebPass_install_dir*\oblix\WebServices\WSDL\um_CanIDeleteUser.wsdl

Parameters:

`uid` The DN of an entry you want to modify.

Rules: Required. Single value, a DN.

Can this user view another user's profile

Function: canUserViewUserProfile

Request example:
```
<oblix:request application="userservcenter"
function="canUserViewUserProfile">
```

Description: Verifies that a non-logged in user can view another user's profile.

WSDL file: *WebPass_install_dir*\oblix\WebServices\WSDL\um_CanUserViewUserProfile.wsdl

Parameters:

`uid` The DN of the user whose profile is to be viewed.

Rules: Required. Single value, a DN.

`proxysourceuid` The DN of a non-logged-in user (proxy user) whose access rights are being tested.

Rules: Required. Single value, a DN.

Can this user view an attribute in another user's profile

Function: canUserViewUserProfileAttr

Request example:
```
<oblix:request application="userservcenter"
function="canUserViewUserProfileAttr">
```

Description: Verifies that a non-logged in user can view a particular attribute in another user's profile.

WSDL file: *WebPass_install_dir*\oblix\WebServices\WSDL\um_CanUserViewUserProfileAttr.wsdl

Parameters:

`uid` The DN for the user whose profile is to be viewed.

Rules: Required. Single value, a DN.

`proxysourceuid` The DN for a non-logged-in user (proxy user) whose access rights are being tested.

Rules: Required. Single value, a DN.

`targetAttribute` The schema name (not the display name) for the desired attribute.

Rules: Required. Single value, a string.

Can this user modify an attribute in another user's profile using a workflow

Function: canUserRequestUserAttrModification

Request example:
```
<oblix:request application="userservcenter"
function="canUserRequestUserAttrModification">
```

Description: Verifies that a user can request a change of an attribute.

WSDL file: *WebPass_install_dir*\oblix\WebServices\WSDL\um_CanUserRequestUserAttrModification.wsdl

Parameters:

`uid` The DN of an entry you want to modify.

Rules: Required. Single value, a DN.

`proxysourceuid` The DN for a non-logged-in user (proxy user) whose access rights are being tested.

Rules: Required. Single value, a DN.

`targetattribute` The schema name (not the display name) for the desired attribute.

Rules: Required. Single value, a string.

Can this user create a new user

Function: canUserCreateUser

Request example:
```
<oblix:request application="userservcenter"
function="canUserCreateUser">
```

Description: Verifies that a non-logged in user can create a new user.

WSDL file: *WebPass_install_dir*\oblix\WebServices\WSDL\um_CanUserCreateUser.wsdl

Parameters:

`proxysourceuid` The DN for a non-logged-in user (proxy user) whose access rights are being tested.

Rules: Required. Single value, a DN.

`ObDomainName` A subtree within which a test is being requested.

Rules: Optional. Single value, a DN.

Default: if no value is provided, NetPoint checks to see if you have the tested rights in *any* domain.

Can this user delete an existing user

Function: canUserDeleteUser

Request example:
```
<oblix:request application="userservcenter"
function="canUserDeleteUser">
```

Description: Verifies that a non-logged in user can delete an existing user.

WSDL file: *WebPass_install_dir*\oblix\WebServices\WSDL\um_CanUserDeleteUser.wsdl

Parameters:

`proxysourceuid` The DN for a non-logged-in user (proxy user) whose access rights are being tested.

Though it is outside the intent of functions using this parameter, DNs other than those of users can be used.

Rules: Required. Single value, a DN.

`uid` The DN of an entry you want to modify.

Rules: Required. Single value, a DN.

Can this user modify another user's attribute

Function: canUserModifyUserProfileAttr

Request example:
```
<oblix:request application="userservcenter"
function="canUserModifyUserProfileAttr">
```

Description: Verifies that a non-logged in user can change a particular attribute in another user's profile.

WSDL file: *WebPass_install_dir*\oblix\WebServices\WSDL\um_CanUserModifyUserProfileAttr.wsdl

Parameters:

`uid` The DN of the user whose attribute you want to modify.

Rules: Required. Single value, a DN.

`proxysourceuid` The DN for a non-logged-in user (proxy user) whose access rights are being tested.

Rules: Required. Single value, a DN.

`targetAttribute` The schema name (not the display name) for the desired attribute.

Rules: Required. Single value, a string.

Can this user request a change to another user's profile using a workflow

Function: canUserRequestUserAttrModification

Request example:
```
<oblix:request application="userservcenter"
function="canUserRequestUserAttrModification">
```

Description: Verifies that a non-logged in user can request a change to a particular attribute in another user's profile, using a workflow.

WSDL file: *WebPass_install_dir*\oblix\WebServices\WSDL\um_CanUserRequestUserAttrModification.wsdl

Parameters:

`uid` The DN of the user whose attribute you want to change.

Rules: Required. Single value, a DN.

`proxysourceuid` The DN for a non-logged-in user (proxy user) whose access rights are being tested.

Though it is outside the intent of functions using this parameter, DNs other than those of users can be used.

Rules: Required. Single value, a DN.

`targetAttribute` The schema name (not the display name) for the desired attribute.

Rules: Required. Single value, a string.

Functions to Perform User Manager Actions

These functions enable you or another user to perform a particular COREid action, such as creating a user. These are get and set functions.

View user attributes

Function: view

Request example:
```
<oblix:request application="userservcenter"
function="view">
```

Description: Use this function to view attributes.

WSDL file: *WebPass_install_dir*\oblix\WebServices\WSDL\um_view.wsdl

Parameters:

`uid`	The DN of the user, in the case of the User Manager. If no uid is specified, the profile of the logged in user will be shown. Rules: Optional for the User Manager only. Single value, a DN. Notes: This parameter also applies to the DN of the group or organization whose attributes are to be viewed, depending upon if this function is being used in the Group Manager or Organization Manager.
`attrName`	Optional. See “Pre-NetPoint 6.5 Attribute Parameters” on page 145 or “NetPoint 6.5 and Higher Attribute Parameters” on page 149. If no attrNames are specified, then all of the attributes of the entry that the logged-in user has access to view are returned. Those attributes must be configured in NetPoint and added to a panel in the User, Group, or Organization Manager.

Note: This function shows deactivated users if the requester is a Master Administrator, or if the administrator has the delegated administration rights of Grant and Workflow Monitoring.

View Example

```
<SOAP-ENV:Envelope xmlns:oblix=vhttp://www.oblix.com"
xmlns:SOAP-ENV="http://schemas-xmlsoap.org/soap/envelope/">
<SOAP-ENV:Body>
   <oblix:authentication xmlns:oblix="http://www.oblix.com" type="basic"?
      <oblix:login>admin</oblix:login>
      <oblix:password>oblix></oblix:password>
   </oblix:authentication>
   <oblix:request application="userservcenter" function="view">
      <oblix:params>
         <oblix:uid>
            cn=test1,o=Company,c=US
         </oblix:uid>
         <oblix:attrName>
            genuserid
         </oblix:attrname>
         <oblix:attrName>
            mail
         </oblix:attrName>
      </oblix:params>
   </oblix:request>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Modify user attributes

Function:	modifyUser
Request example:	`<oblix:request application="userservcenter" function="modifyUser">`
Description:	Change the attribute values for a specified user.
WSDL file:	*WebPass_install_dir*\oblix\WebServices\WSDL\ um_modifyUser.wsdl
Parameters:	
`uid`	The DN of the user whose attributes are to be changed. Rules: Required. Single value, a DN.
`attrName`	Optional. See “Pre-NetPoint 6.5 Attribute Parameters” on page 145 or “NetPoint 6.5 and Higher Attribute Parameters” on page 149. Without the _n, to return data for only the named attributes. Though optional for this function, it is best to always provide this parameter. The trade-off is that if you omit it, you get back data for all the names that appear in the panel. Use this parameter to limit output to just the data you want to see. You use this parameter in addition to the attrName_n parameter.
`attrName_n`	Required (old syntax). See “Pre-NetPoint 6.5 Attribute Parameters” on page 145.
`attrName`	Required (new syntax). See “NetPoint 6.5 and Higher Attribute Parameters” on page 149.
`attrOperation_n`	Required (old syntax). See “Pre-NetPoint 6.5 Attribute Parameters” on page 145.
`attrOperation`	Required (new syntax). See “NetPoint 6.5 and Higher Attribute Parameters” on page 149.
`attrValue_n`	Required (old syntax). See “Pre-NetPoint 6.5 Attribute Parameters” on page 145.

`attrNewValue` Required (new syntax). See “NetPoint 6.5 and Higher Attribute Parameters” on page 149.

`NoOfFields` Required. See “Pre-NetPoint 6.5 Attribute Parameters” on page 145 or “NetPoint 6.5 and Higher Attribute Parameters” on page 149.

`attrOldValue_n` Optional/Required (old syntax). Required only if the attrOperation is a REPLACE. See “Pre-NetPoint 6.5 Attribute Parameters” on page 145.

`attrOldValue` Optional/Required (new syntax). Required only if the attrOperation is a REPLACE. See “NetPoint 6.5 and Higher Attribute Parameters” on page 149.

Request user attribute change through a workflow

Function: workflowSaveChangeAttributeRequest

Request example:

```
<oblix:request application="userservcenter"
function="workflowSaveChangeAttributeRequest">
```

Description: Use this function to request a group, organization, or user attribute change using a workflow. The parameters starting with OBAuxClasses apply only to groups.

WSDL file: *WebPass_install_dir*\oblix\WebServices\WSDL\um_workflowSaveChangeAttributeRequest.wsdl

Parameters:

`uid` The DN of the user, group or organization whose attribute is to be changed.

Rules: Required. Single value, a DN.

`attrName_n` Required (old syntax). Required here means attributes that are specific to each workflow. See “Pre-NetPoint 6.5 Attribute Parameters” on page 145.

`attrName` Required (new syntax). Required here means attributes that are specific to each workflow. If an attribute supplied here is not required by the workflow, it is ignored, and no error is generated. See “NetPoint 6.5 and Higher Attribute Parameters” on page 149.

`attrOperation_n` Required (old syntax). See “Pre-NetPoint 6.5 Attribute Parameters” on page 145.

`attrOperation` Required (new syntax). See “NetPoint 6.5 and Higher Attribute Parameters” on page 149.

`attrValue_n` Required (old syntax). See “Pre-NetPoint 6.5 Attribute Parameters” on page 145.

`attrNewValue` Required (new syntax). See “NetPoint 6.5 and Higher Attribute Parameters” on page 149.

`changeRequestAttr`

Use this parameter to name the attribute whose value you want to change. This is the LDAP schema name of the attribute, not the display name.

Rules: Required. Single-valued, a string.

`changeRequestType`

Specifies whether this request is a provisioning or deprovisioning request.

Rules: Required. Single value. It can be one of two values: remove (for deprovisioning) newval (for provisioning).

`NoOfFields` Required. See “Pre-NetPoint 6.5 Attribute Parameters” on page 145 or “NetPoint 6.5 and Higher Attribute Parameters” on page 149.

`ObWorkflowName` The name of the workflow that you want to use to create or change the value(s) for an attribute.

Find the full DN for ObWorkflowName under the view menu for workflow definition under the particular application.

Rules: Required. Single value, a DN.

`attrOldValue_n` Optional/Required (old syntax). Required only if the attrOperation is a REPLACE. See “Pre-NetPoint 6.5 Attribute Parameters” on page 145.

`attrOldValue` Optional/Required (new syntax). Required only if the attrOperation is a REPLACE.

`ObAuxClassesOldValues`

The old values of the auxiliary class names that you want to replace. This is used only to change the name information for auxiliary classes associated with groups. Use this parameter once for each auxiliary class name to be removed.

If you attempt to specify a value for which you do not have access, you will get an error message "Invalid value for attributeObAuxClasses."

You find the values for these using Group Manager > Group Manager Configuration > Configure Group Types > Configure Group Type Panels. Select the group, and find the Associated ObjectClass name displayed.

Rules: Required only if the attribute is for an auxiliary class and the ObAuxClassesOperation is a REPLACE, otherwise ignored. Multivalued.

`ObAuxClassesOperation`

The type of operation to perform on the attribute. This is used only to change the name information for auxiliary classes.

Legal values are:

`ADD`—Add the auxiliary class name to the existing attributes.

`DELETE`—Delete the auxiliary class name from the existing attributes.

`REPLACE`—Delete the old auxiliary class name and replace it with the new auxiliary class name.

If you specify any other value or no value, you will get an error message "Invalid value for attribute ObAuxClasses."

Rules: Required only if the attribute is for an auxiliary class. Single value.

`ObAuxClassesValues`

The name of the auxiliary class that you want to add, delete, or replace. This is used only to change the name information for auxiliary classes.

Use this parameter once for each auxiliary class name to be added or removed.If you attempt to specify a value for which you do not have access, you will get an error message "Invalid value for attributeObAuxClasses".

To find the values for these, click Group Manager > Group Manager Configuration > Configure Group Types > Configure Group Type Panels. Select the group, and find the Associated ObjectClass name.

Rules: Required if the attribute is for an auxiliary class. Multivalued. Valid values are the string names of the configured auxiliary classes available. (Auxiliary classes are configured through the System Console, Configure Object Class function, see the *NetPoint 7.0 Administration Guide Volume 1*.)

`ObWfComment` Use this parameter to provide a comment for a step in a workflow.

Rules: Optional. Single value, string.

Create User

Function: workflowSaveCreateProfile

Request example:
```
<oblix:request application="userservcenter"
function="workflowSaveCreateProfile">
```

Description: Use this function to create a new user, group, or organization using a workflow.The parameters starting with OBAuxClasses apply only to groups.

WSDL file: *WebPass_install_dir*\oblix\WebServices\WSDL\ um_workflowSaveCreateProfile.wsdl

Parameters:

`ObDomainName` The name of the domain where you want to create a new entry.

Rules: Required. Single value, a DN. The domain name must be defined under the workflow referred to by the ObWorkflowName parameter.

`ObWorkflowName` The name of the workflow that you want to use to create or change the value(s) for an attribute.

Find the full DN for ObWorkflowName under the view menu for workflow definition under the particular application.

Rules: Required. Single value, a DN.

`NoOfFields` Required. See “Pre-NetPoint 6.5 Attribute Parameters” on page 145 or “NetPoint 6.5 and Higher Attribute Parameters” on page 149.

`attrName_n` Required (old syntax). Required here means attributes that are specific to each workflow. See “Pre-NetPoint 6.5 Attribute Parameters” on page 145.

`attrName` Required (new syntax). Required here means attributes that are specific to each workflow. If an attribute supplied here is not required by the workflow, it is ignored, and no error is generated. See “NetPoint 6.5 and Higher Attribute Parameters” on page 149.

`attrOperation_n` Required (old syntax). See “Pre-NetPoint 6.5 Attribute Parameters” on page 145.

`attrOperation` Required (new syntax). See “NetPoint 6.5 and Higher Attribute Parameters” on page 149.

`attrValue_n` Required (old syntax). See “Pre-NetPoint 6.5 Attribute Parameters” on page 145.

`attrNewValue` Required (new syntax). Required. See “NetPoint 6.5 and Higher Attribute Parameters” on page 149.

`attrOldValue` Optional/Required. Required only if the attrOperation is a REPLACE.

`ObAuxClassesOldValues`
The old values of the auxiliary class names that you want to replace. This is used only to change the name information for auxiliary classes associated with groups. Use this parameter once for each auxiliary class name to be removed.

If you attempt to specify a value for which you do not have access, you will get an error message "Invalid value for attributeObAuxClasses."

You find the values for these using Group Manager > Group Manager Configuration > Configure Group Types > Configure Group Type Panels. Select the group, and find the Associated ObjectClass name displayed.

Rules: Required only if the attribute is for an auxiliary class and the ObAuxClassesOperation is a REPLACE, otherwise ignored. Multivalued.

`ObAuxClassesOperation`

The type of operation to perform on the attribute. This is used only to change the name information for auxiliary classes.

Legal values are:

`ADD`—Add the auxiliary class name to the existing attributes.

`DELETE`—Delete the auxiliary class name from the existing attributes.

`REPLACE`—Delete the old auxiliary class name and replace it with the new auxiliary class name.

If you specify any other value or no value, you will get an error message "Invalid value for attribute ObAuxClasses."

Rules: Required if the attribute is for an auxiliary class. Single value.

`ObAuxClassesValues`

The name of the auxiliary class to add, delete, or replace. This is used only to change the name information for auxiliary classes.

Use this parameter once for each auxiliary class name to be added or removed.If you attempt to specify a value for which you do not have access, you get an error message "Invalid value for attributeObAuxClasses."

To find the values, click Group Manager > Group Manager Configuration > Configure Group Types > Configure Group Type Panels. Select the group, and find the Associated ObjectClass name.

Rules: Required if the attribute is for an auxiliary class. Multivalued. Valid values are the string names of the configured auxiliary classes available. Auxiliary classes are configured through the Administration Console's configure object classes function. See the *NetPoint 7.0 Administration Guide Volume 1* for details.

`ObWfComment` Provides a comment for a step in a workflow.

Rules: Optional. Single value, string.

Self-Registration

Function: workflowSelfRegistrationSave

Request example:
```
<oblix:request application="userservcenter"
function="workflowSelfRegistrationSave">
```

Description: Adds yourself to an organization or as a user.

WSDL file: *WebPass_install_dir*\oblix\WebServices\WSDL\um_workflowSelfRegistrationSave.wsdl

Parameters:

`ObDomainName` The name of the domain in which you want to create a new entry. The domain name must be defined under the workflow referred to by the ObWorkflowName parameter.

Rules: Required. Single value, a DN.

`ObWorkflowName` The name of the workflow that you want to use to create or change the value(s) for an attribute.

Find the full DN for ObWorkflowName under the view menu for workflow definition under the particular application.

Rules: Required. Single value, a DN.

`attrName_n`	Required (old syntax). Required here means attributes that are specific to each workflow. See “Pre-NetPoint 6.5 Attribute Parameters” on page 145.
`attrName`	Required (new syntax). Required here means attributes that are specific to each workflow. If an attribute supplied here is not required by the workflow, it is ignored, and no error is generated. See “NetPoint 6.5 and Higher Attribute Parameters” on page 149.
`attrOperation_n`	Required (old syntax). See “Pre-NetPoint 6.5 Attribute Parameters” on page 145.
`attrOperation`	Required (new syntax). See “NetPoint 6.5 and Higher Attribute Parameters” on page 149.
`attrValue_n`	Required (old syntax). See “Pre-NetPoint 6.5 Attribute Parameters” on page 145.
`attrNewValue`	Required (new syntax). Required. See “NetPoint 6.5 and Higher Attribute Parameters” on page 149.
`NoOfFields`	Required. See “Pre-NetPoint 6.5 Attribute Parameters” on page 145 or “NetPoint 6.5 and Higher Attribute Parameters” on page 149.
`attrOldValue`	Optional/Required. Required only if the attrOperation is a REPLACE.
`ObWfComment`	Provides a comment for a step in a workflow. Rules: Optional. Single value, string.

Deactivate User

Function:	workflowDeactivateUserSave
Request example:	`<oblix:request application="userservcenter" function="workflowDeactivateUserSave">`
Description:	Deactivates a user using a workflow. Information for deactivated users is kept in the directory but not shown in search results.

WSDL file: *WebPass_install_dir*\oblix\WebServices\WSDL\um_workflowDeactivateUserSave.wsdl

Parameters:

`uid`	The DN of the user to be deactivated. Rules: Required. Single value, a DN.
`ObDomainName`	The name of the domain in which you want to create a new entry. The domain name must be defined under the workflow referred to by the ObWorkflowName parameter. Rules: Required. Single value, a DN.
`ObWorkflowName`	The name of the workflow that you want to use to create or change the value(s) for an attribute. Find the full DN for ObWorkflowName under the view menu for workflow definition under the particular application. Rules: Required. Single value, a DN.
`attrName_n`	Required (old syntax). Here, required means attributes that are specific to each workflow. See “Pre-NetPoint 6.5 Attribute Parameters” on page 145.
`attrName`	Required (new syntax). Here, required means attributes that are specific to each workflow. If an attribute supplied here is not required by the workflow, it is ignored, and no error is generated. See “NetPoint 6.5 and Higher Attribute Parameters” on page 149.
`attrOperation_n`	Required (old syntax). See “Pre-NetPoint 6.5 Attribute Parameters” on page 145.
`attrOperation`	Required (new syntax). See “NetPoint 6.5 and Higher Attribute Parameters” on page 149.
`attrValue_n`	Required (old syntax). See “Pre-NetPoint 6.5 Attribute Parameters” on page 145.
`attrNewValue`	Required (new syntax). See “NetPoint 6.5 and Higher Attribute Parameters” on page 149.

`NoOfFields`	Required. Required. See “Pre-NetPoint 6.5 Attribute Parameters” on page 145 or “NetPoint 6.5 and Higher Attribute Parameters” on page 149.
`attrOldValue`	Optional/Required. Required only if the attrOperation is a REPLACE.
`ObWfComment`	Provides a comment for a step in a workflow. Rules: Optional. Single value, string.

View Deactivated User

To be able to view a deactivated user, you need to be either the master administrator or have delegated administration rights for grant and workflow monitoring of the domain where the target user exists. That is, you should be able to monitor workflow requests in the target user domain. Then you can view a deactivated user in that domain using the view function. See “View user attributes” on page 93.

Search Deactivated Users

Function: searchDeactivatedUsers

Request example:
```
<oblix:request application="userservcenter"
function="searchDeactivatedUsers">
```

Description: Search for deactivated users, based on certain criteria. Only one search condition is accepted. You can search for deactivated users based on one condition only.

To be able to search for deactivated users, you need to be either the master administrator or have delegated administration rights to GRANT+WORKFLOW MONITORING to the domain where the target users exist. That is, you should be able to monitor workflow requests in the target users' domain. Then you can search for deactivated users in that domain using the searchDeactivatedUsers function. One difference is that the result attributes are those specified in the search results table so you cannot specify the result attributes through attrName as you can do in a normal search.

WSDL file: *WebPass_install_dir*\oblix\WebServices\WSDL\um_searchDeactivatedUsers.wsdl

Parameters:

`SLkn`	Required (old syntax). See “Pre-NetPoint 6.5 Search Parameters” on page 140.
`SStn`	Required (old syntax). See “Pre-NetPoint 6.5 Search Parameters” on page 140.
`STyn`	Required (old syntax). See “Pre-NetPoint 6.5 Search Parameters” on page 140.
`SearchAttr`	Required (new syntax). See “NetPoint 6.5 Search Parameters” on page 142.
`SearchOperation`	Required (new syntax). See “NetPoint 6.5 Search Parameters” on page 142.
`SearchString`	Required (new syntax). See “NetPoint 6.5 Search Parameters” on page 142.
`attrname`	Optional. If no value is given, the default table view attributes are used See “Pre-NetPoint 6.5 Attribute Parameters” on page 145 or “NetPoint 6.5 and Higher Attribute Parameters” on page 149.
`noOfFields`	Optional. See “Pre-NetPoint 6.5 Attribute Parameters” on page 145 or “NetPoint 6.5 and Higher Attribute Parameters” on page 149.
`noOfRecords`	Optional. A maximum number of entries to be returned in the search results.This, and its default value, is overridden by the showAllResults parameter. Rules: Optional. Single value, an integer value >=1. Default: A value obtained from the defaultDisplayResultVal parameter in the oblixbaseparams.xml catalog. Otherwise this value is obtained from the custom cookie.
`showAllResults`	Returns all results of the search to the user. If the parameter value is true, it overrides the value of the noOfRecords parameter. Rules: Optional. Single value, Boolean, valued true or false.

Default: false, meaning return results up to the limit imposed by the noOfRecords parameter.

`sortBy` — What attribute to use to sort the results.

Rules: Optional. Single value.

Default: if no value is specified, the class attribute of the structural objectclass of the tab specified by tab_id is used.

`sortOrder` — The sort order, ascending or descending. There are two possible values: ascending, descending.

Rules: Optional. Single value.

Default: ascending

`startFrom` — Use this parameter for a long list of search results, to skip over a selected number of items and start the list with a specified item. For example, if 100 entries were found by the search, entering a value of 80 for this parameter gives a response showing only items 80 through 100.

Rules: Optional. Single value, integer.

Default: 1, meaning to start displaying from the beginning of the search results list.

Group Manager Functions

If you are an administrator, the Group Manager enables you to create or delete groups, and enables users to subscribe or unsubscribe from groups. The Group Manager typically enables end users to view groups and to subscribe to membership in a group. The groups that a person can view and subscription rights are granted by a NetPoint administrator.

The following functions allow you to programmatically access the Group Manager application. Note that all functions follow a similar syntax:

<oblix:request application="groupservcenter" function="*name*">

For example:

<oblix:request application="groupservcenter"
function="canIViewGroupProfile">

Functions to Test for Attribute Permissions

The following functions provide a yes or no response as to whether you or another user have read, write, delegate, and notify permissions set for a particular attribute.

Can I view a group's profile

Function: canIViewGroupProfile

Request example: `<oblix:request application="groupservcenter" function="canIViewGroupProfile">`

Description: Verifies that you can view a group's profile.

WSDL file: *WebPass_install_dir*\oblix\WebServices\WSDL\gm_**canIViewGroupProfile**.wsdl

Parameters:

`uid` — The DN of the group whose profile you want to view.

Rules: Required. Single value, a DN.

Example

```
<?xml version="1.0"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas-xmlsoap.org/soap/envelope/">
<SOAP-ENV:Body>
   <oblix:request application="groupservcenter"
      function="canIViewGroupProfile">
   <oblix:params>
      <oblix:param name="uid">cn=Marketing Team, ou=Marketing, o=Company, c=US
      </oblix:param>
   </oblix:params>
   </oblix:request>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Can I view an attribute in a group's profile

Function: canIViewGroupProfileAttr

Request example: `<oblix:request application="groupservcenter" function="canIViewGroupProfileAttr">`

Description: Verifies that you can view a particular attribute in a group's profile.

WSDL file: *WebPass_install_dir*\oblix\WebServices\WSDL\gm_**canIViewGroupProfileAttr**.wsdl

Parameters:

`uid` The DN of the group whose attribute you want to view.

Rules: Required. Single value, a DN.

`targetAttribute` The schema name (not the display name) for the desired attribute.

Rules: Required. Single value, a string.

Can I modify an attribute in a group's profile

Function: canIModifyGroupProfileAttr

Request example:
```
<oblix:request application="groupservcenter"
function="canIModifyGroupProfileAttr">
```

Description: Verifies that you can change a particular attribute in a group's profile.

WSDL file: *WebPass_install_dir*\oblix\WebServices\WSDL\gm_canIModifyGroupProfileAttr.wsdl

Parameters:

`uid` The DN of the group whose attribute you want to change.

Rules: Required. Single value, a DN.

`targetAttribute` The schema name (not the display name) for the desired attribute.

Rules: Required. Single value, a string.

Can I request modification through a workflow of an attribute in a group profile

Function: canIRequestGroupAttrModification

Request example:
```
<oblix:request application="groupservcenter"
function="canIRequestGroupAttrModification">
```

Description: Verifies that you can change a particular attribute in a group's profile, using a workflow.

WSDL file: *WebPass_install_dir*\oblix\WebServices\WSDL\gm_**canIRequestGroupAttrModification**.wsdl

Parameters:

`uid` The DN of the group whose attribute you want to change.

Rules: Required. Single value, a DN.

`targetAttribute` The schema name (not the display name) for the desired attribute.

Rules: Required. Single value, a string.

Can I create a new group

Function: canICreateGroup

Request example:
```
<oblix:request application="groupservcenter"
function="canICreateGroup">
```

Description: Verifies that you can create a new group.

WSDL file: *WebPass_install_dir*\oblix\WebServices\WSDL\gm_**canICreateGroup**.wsdl

Parameters:

`ObDomainName` A subtree within which a test is being requested.

Rules: Optional. Single value, a DN.

Default: if no value is provided, NetPoint checks to see if you have the tested rights in *any* domain.

`Objectclass` The auxiliary object class(es), if any, within which the group is to be created. This applies only to Group Manager, where the auxiliary objectclasses correspond to the group types.

You find the values for these using Group Manager > Group Manager Configuration > Configure Group Types > Configure Group Type Panels. Select the group, and find the Associated ObjectClass name displayed.

Rules: Optional. Multivalued.

Can I delete an existing group

Function: canIDeleteGroup

Request example:
```
<oblix:request application="groupservcenter"
function="canIDeleteGroup">
```

Description: Verifies that you can delete an existing group.

WSDL file: *WebPass_install_dir*\oblix\WebServices\WSDL\gm_**canIDeleteGroup**.wsdl

Parameters:

`uid` The DN of an entry you want to modify.

Rules: Required. Single value, a DN.

Can I subscribe to a group

Function: canISubscribeToGroup

Request example:
```
<oblix:request application="groupservcenter"
function="canISubscribetoGroup">
```

Description: Verifies that you can subscribe to a specific group.

WSDL file: *WebPass_install_dir*\oblix\WebServices\WSDL\gm_**canISubscribeToGroup**.wsdl

Parameters:

`uid` The DN of the group to which you want to subscribe.

Rules: Required. Single value, a DN.

Can I unsubscribe from a group

Function: canIUnSubscribeFromGroup

Request example:
```
<oblix:request application="groupservcenter"
function="canIUnSubscribeFromGroup">
```

Description: Verifies that you can unsubscribe from a specific group.

WSDL file: *WebPass_install_dir*\oblix\WebServices\WSDL\gm_**canIUnSubscribeFromGroup**.wsdl

Parameters:

`uid` The DN of the group from which you want to unsubscribe.

Rules: Required. Single value, a DN.

Am I a member of a group

Function: amIAMember

Request example:
```
<oblix:request application="groupservcenter"
function="amIAMember">
```

Description: Use this function to determine if the logged in user is a member of any group. It checks for static membership by default. If you also want to test the nested or dynamic membership, you need to use the optional flags as described below. Use the function memberOfAGroup to determine third-person group membership.

WSDL file: *WebPass_install_dir*\oblix\WebServices\WSDL\gm_**amIAMember**.wsdl

Parameters:

`uid` The DN of the entry you want to query.

Rules: Required. Single value, a DN.

`checkNested` Set this parameter to true to check nested groups for membership.

Rules: Optional. Single-valued, a flag.

Default:false.

`checkDynamic` Set this parameter to true to check dynamic groups for membership.

Rules: Optional. Single-valued, a flag.

Default: false.

Can a user view a group's profile

Function: canUserViewGroupProfile

Request example:
```
<oblix:request application="groupservcenter"
function="canUserViewGroupProfile">
```

Description: Verifies that a non-logged in user can view a group's profile.

WSDL file: *WebPass_install_dir*\oblix\WebServices\WSDL\gm_**canUserViewGroupProfile**.wsdl

Parameters:

`uid` The DN of the group whose profile you want to view.

Rules: Required. Single value, a DN.

`proxysourceuid` The DN for a non-logged-in user (proxy user) whose access rights are being tested.

Though it is outside the intent of functions using this parameter, DNs other than those of users can be used.

Rules: Required. Single value, a DN.

Can a user view an attribute in a group's profile

Function: canUserViewGroupProfileAttr

Request example:
```
<oblix:request application="groupservcenter"
function="canUserViewGroupProfileAttr">
```

Description: Verifies that a non-logged in user can view a particular attribute in a group's profile.

WSDL file: *WebPass_install_dir*\oblix\WebServices\WSDL\gm_**canUserViewGroupProfileAttr**.wsdl

Parameters:

`uid` The DN of the group whose attribute you want to view.

Rules: Required. Single value, a DN.

`proxysourceuid` The DN for a non-logged-in user (proxy user) whose access rights are being tested.

Though it is outside the intent of functions using this parameter, DNs other than those of users can be used.

Rules: Required. Single value, a DN.

`targetAttribute` The schema name (not the display name) for the desired attribute.

Rules: Required. Single value, a string.

Can a user modify an attribute in a group profile using a workflow

Function: canUserRequestGroupAttrModification

Request example:

```
<oblix:request application="groupservcenter"
function="canUserRequestGroupAttrModification">
```

Description: Verifies that a non-logged in user can request a change to a particular attribute in a group's profile, using a workflow.

WSDL file: *WebPass_install_dir*\oblix\WebServices\WSDL\gm_**canUserRequestGroupAttrModification**.wsdl

Parameters:

`uid` The DN of the group whose attribute you want to modify.

Rules: Required. Single value, a DN.

`proxysourceuid` The DN for a user (proxy user) whose access rights are being tested.

Rules: Required. Single value, a DN.

`targetAttribute` The schema name (not the display name) for the desired attribute.

Rules: Required. Single value, a string.

Can a user create a new group

Function: canUserCreateGroup

Request example:
```
<oblix:request application="groupservcenter"
function="canUserCreateGroup">
```

Description: Verifies that a non-logged in user can create a new group.

WSDL file: *WebPass_install_dir*\oblix\WebServices\WSDL\gm_**canUserCreateGroup**.wsdl

Parameters:

`proxysourceuid` The DN for a user (proxy user) whose rights are being tested.

Rules: Required. Single value, a DN.

`ObDomainName` A subtree within which a test is being requested.

Rules: Optional. Single value, a DN.

Default: if no value is provided, NetPoint checks to see if you have the tested rights in *any* domain.

`Objectclass` The auxiliary object class(es), if any, within which the group is to be created. This applies only to Group Manager, where the auxiliary objectclasses correspond to the group types.

You find the values for these using Group Manager > Group Manager Configuration > Configure Group Types > Configure Group Type Panels. Select the group, and find the Associated ObjectClass name displayed.

Rules: Optional. Multivalued.

Can a user delete an existing group

Function: canUserDeleteGroup

Request example:
```
<oblix:request application="groupservcenter"
function="canUserDeleteGroup">
```

Description: Verifies that a non-logged in user can delete an existing group.

WSDL file: *WebPass_install_dir*\oblix\WebServices\WSDL\gm_**canUserDeleteGroup**.wsdl

Parameters:

`proxysourceuid` The DN for a non-logged-in user (proxy user) whose access rights are being tested.

Rules: Required. Single value, a DN.

`uid` The DN of the entry.

Rules: Optional. Single value, a DN.

Is this person a member of a group

Function: memberOfAGroup

Request example:
```
<oblix:request application="groupservcenter"
function="memberOfAGroup">
```

Description: Third-person IdentityXML request to check a person's membership to a particular group. It checks for static membership by default. If you also want to test the nested or dynamic membership, you need to use the optional flags as described below. You will need to have view access for the dynamic filter attribute.

WSDL file: *WebPass_install_dir*\oblix\WebServices\WSDL\gm_**memberOfAGroup**.wsdl

Parameters:

`proxysourceuid` The DN for a non-logged-in user (proxy user) whose access rights are being tested.

Rules: Required. Single value, a DN.

`uid` The DN of the entry.

Rules: Required. Single value, a DN.

`checkNested` Set this parameter to true to check nested groups for membership.

Rules: Optional. Single-valued, a flag.

Default:false

`checkDynamic` Set this parameter to true to check dynamic groups for membership.

Rules: Optional. Single-valued, a flag.

Default: false.

Request group attribute change

Function: canUserModifyGroupProfileAttr

Request example:
```
<oblix:request application="groupservcenter"
function="canUserModifyGroupProfileAttr">
```

Description: Verifies that a non-logged in user can change a particular attribute in a group's profile.

WSDL file: *WebPass_install_dir*\oblix\WebServices\WSDL\gm_**canUserModifyGroupProfileAttr**.wsdl

Parameters:

`uid` The DN of the group whose attribute you want to modify.

Rules: Required. Single value, a DN.

`proxysourceuid` The DN for a non-logged-in user (proxy user) whose access rights are being tested.

Rules: Required. Single value, a DN.

`targetAttribute` The schema name (not the display name) for the desired attribute.

Rules: Required. Single value, a string.

Request group attribute change through a workflow

See “Request user attribute change through a workflow” on page 96. Note that the application name is groupservcenter.

Functions to Perform Group Manager Actions

The following IdentityXML functions allow you to perform the following actions. These are get and set functions.

View group attributes

See “View user attributes” on page 93. Note that the application name is groupservcenter.

Modify Group attributes

Function: modifyGroup

Request example:
```
<oblix:request application="groupservcenter"
function="modifyGroup">
```

Request example:
```
<oblix:request application="groupservcenter"
function="modifyGroup">
```

Description: Use this function to change group attributes.

WSDL file: *WebPass_install_dir*\oblix\WebServices\WSDL\gm_**modifyGroup**.wsdl

Parameters:

`uid` The DN of the group whose attributes are to be changed.

Rules: Required. Single value, a DN.

`attrName` Optional (old syntax). Without the _n (old syntax), to return data for only the named attributes. Though optional for this function, it is best to always provide this parameter. The trade-off is that if you omit it, you get back data for all the names that appear in the panel. Use this parameter to limit output to just the data you want to see. You use this parameter in addition to the attrName_n parameter.

`attrName_n` Required (old syntax). Optional. If no value is given, the default table view attributes are used. See “Pre-NetPoint 6.5 Attribute Parameters” on page 145.

`attrName` Required (new syntax). See “NetPoint 6.5 and Higher Attribute Parameters” on page 149.

`attrOperation_n` Required (old syntax). See “Pre-NetPoint 6.5 Attribute Parameters” on page 145.

`attrOperation` Required (new syntax). See “NetPoint 6.5 and Higher Attribute Parameters” on page 149.

`attrValue_n` Required (old syntax). See “Pre-NetPoint 6.5 Attribute Parameters” on page 145.

`attrNewValue` Required (new syntax). Required. See “NetPoint 6.5 and Higher Attribute Parameters” on page 149.

`noOfFields` Required. See “Pre-NetPoint 6.5 Attribute Parameters” on page 145 or “NetPoint 6.5 and Higher Attribute Parameters” on page 149.

`ObAuxClassesOldValues`
The old values of the auxiliary class names that you want to replace. This is used only to change the name information for auxiliary classes associated with groups. Use this parameter once for each auxiliary class name to be removed.

If you attempt to specify a value for which you do not have access, you will get an error message "Invalid value for attributeObAuxClasses."

You find the values for these using Group Manager > Group Manager Configuration > Configure Group Types > Configure Group Type Panels. Select the group, and find the Associated ObjectClass name displayed.

Rules: Required only if the attribute is for an auxiliary class and the ObAuxClassesOperation is a REPLACE, otherwise ignored. Multivalued.

`ObAuxClassesOperation`

The type of operation to perform on the attribute. This is used only to change the name information for auxiliary classes.

Legal values are:

ADD—Add the auxiliary class name to the existing attributes.

DELETE—Delete the auxiliary class name from the existing attributes.

REPLACE—Delete the old auxiliary class name and replace it with the new auxiliary class name.

If you specify any other value or no value, you will get an error message "Invalid value for attribute ObAuxClasses."

Rules: Required only if the attribute is for an auxiliary class. Single value.

`ObAuxClassesValues`

The name of the auxiliary class that you want to add, delete, or replace. This is used only to change the name information for auxiliary classes.

Use this parameter once for each auxiliary class name to be added or removed.If you attempt to specify a value for which you do not have access, you will get an error message "Invalid value for attributeObAuxClasses".

To find the values for these, click Group Manager > Group Manager Configuration > Configure Group Types > Configure Group Type Panels. Select the group, and find the Associated ObjectClass name.

Rules: Required if the attribute is for an auxiliary class. Multivalued. Valid values are the string names of the configured auxiliary classes available. (Auxiliary classes are configured through the System Console Configure Object Class function, see the *NetPoint 7.0 Administration Guide Volume 1*.)

`attrOldValue_n` Optional/Required (old syntax). Required only if the attrOperation is a REPLACE. See “Pre-NetPoint 6.5 Attribute Parameters” on page 145.

`attrOldValue`—Optional/Required (new syntax). Required only if the attrOperation is a REPLACE. This needs to be an exact match. If not, no change takes place.

See “NetPoint 6.5 and Higher Attribute Parameters” on page 149.

Create group

See “Create User” on page 99. Note that the application name is groupservcenter.

Delete Group

Function: delete (group)

Request example:
```
<oblix:request application="groupservcenter"
function="delete">
```

Description: Use this function to delete a group. You have very little control over this function, beyond specifying the name of the group. Determination of the workflow to be used is made by the application. The workflow selected satisfies all of the following: 1) the target domain contains the group entry; 2) you are a participant in the initiate step of the workflow, and 3) the group types of the group to be deleted are a subset of the group types in the workflow definition.

WSDL file: *WebPass_install_dir*\oblix\WebServices\WSDL\gm_**delete**.wsdl

Parameters:

`uid` Rules: Required. Single value, a DN.

Get groups that I am a member, owner, or administrator of

Function: myGroupsProfile

Request example:
```
<oblix:request application="groupservcenter"
function="myGroupsProfile">
```

Description: Use this function to get the profiles for groups you are a member, owner or an administrator of. Parameters used here override the configured Group Manager Options.

Note: The show... options do not all have the same precedence. The parameters showOwnerOfGroups and showAdministratorOfGroups will always be applied if entered. The parameter showMemberOfGroups must be set to true to use showStaticGroups, showDynamicGroups, and showNestedGroups. The showNestedGroups parameter can only be set to true if either or both of the nested group categories showStaticGroups and showDynamicGroups is set to true.The IdentityXML request uses these options instead of the configured Group Manager options that may have been set using the Administration Console.

WSDL file: *WebPass_install_dir*\oblix\WebServices\WSDL\gm_**myGroupsProfile**.wsdl

Parameters:

`attrName` Optional. See “Pre-NetPoint 6.5 Attribute Parameters” on page 145 or “NetPoint 6.5 and Higher Attribute Parameters” on page 149.

`showAdministratorOfGroups`
Use this parameter to ask for groups you, or another user, are an administrator of to be included in the response.

Rules: Optional. At least one of the parameters in the showxxx list must be used. Single value, Boolean, true or false.

Default: false

`showDynamicGroups`
Use this parameter to ask for groups you, or another user, are a dynamic member of to be included in the response.

showMemberOfGroups must also be used, set to true.

Rules: Optional. Single value, Boolean, true or false.

Default: false.

`showMemberOfGroups`
Use this parameter to ask for groups that you, or another user, are a member of to be included in the response.

Rules: Optional. Single value, Boolean, true or false.

Default: false—Optional.

`showNestedGroups` Use this parameter to ask for nested groups you, or another user, are a member of to be included in the response. showMemberOfGroups must also be used, set to true. And one or both of showStaticGroups and showDynamicGroups must also be used, set to true.

Rules: Optional. Single value, Boolean, true or false.

Default: false

`showOwnerOfGroups`Use this parameter to ask for groups you, or another user, are an owner of to be included in the output.

Rules: Optional. Single value, Boolean, true or false.

Default: false

`showStaticGroups` Use this parameter to ask for groups you, or another user, are a static member of to be included in the response.

showMemberOfGroups must also be used, set to true.

Rules: Optional. Single value, Boolean, true or false.

Default: false

Get groups that a user is a member, owner, or administrator of

Function: userGroupsProfile

Request example:
```
<oblix:request application="groupservcenter"
function="userGroupsProfile">
```

Description: Use this function to get the profiles for groups that another user is a member, owner or an administrator of.

Rights: The logged-in user must be able to grant read capability on the proxysourceuid classname attribute.

WSDL file: *WebPass_install_dir*\oblix\WebServices\WSDL\gm_**userGroupsProfile**.wsdl

Parameters:

`proxysourceuid` The DN for a non-logged-in user (proxy user) whose group profile you want.

Rules: Required. Single value, a DN.

`attrName` Optional. See "Pre-NetPoint 6.5 Attribute Parameters" on page 145 or "NetPoint 6.5 and Higher Attribute Parameters" on page 149.

`showAdministratorOfGroups`
Use this parameter to ask for groups you, or another user, are an administrator of to be included in the response.

Rules: Optional. At least one of the parameters in the showxxx list must be used. Single value, Boolean, true or false.

Default: false

`showDynamicGroups`
Use this parameter to ask for groups you, or another user, are a dynamic member of to be included in the response.

showMemberOfGroups must also be used, set to true.

Rules: Optional. Single value, Boolean, true or false.

Default: false.

`showMemberOfGroups`
Use this parameter to ask for groups that you, or another user, are a member of to be included in the response.

Rules: Optional. Single value, Boolean, true or false.

Default: false—Optional.

`showNestedGroups` Use this parameter to ask for nested groups you, or another user, are a member of to be included in the response. showMemberOfGroups must also be used, set to true. And one or both of showStaticGroups and showDynamicGroups must also be used, set to true.

Rules: Optional. Single value, Boolean, true or false.

Default: false

`showOwnerOfGroups` Use this parameter to ask for groups you, or another user, are an owner of to be included in the output.

Rules: Optional. Single value, Boolean, true or false.

Default: false

`showStaticGroups` Use this parameter to ask for groups you, or another user, are a static member of to be included in the response.

showMemberOfGroups must also be used, set to true.

Rules: Optional. Single value, Boolean, true or false.

Default: false

View group members

Function: viewGroupMembers

Request example:

```
<oblix:request application="groupservcenter"
function="viewGroupMembers">
```

Description: View all or selected members of a group. To get selected members of the group, you use the SLkn, SStn, and STyn parameters (old syntax), or the SearchAttr, SearchOperation, or SearchString parameters. You may use only one set. The length of the string value provided for SStn or SearchString must be at greater than or equal to the value for the groupMemberSearchStringMiminumLength parameter in the groupservcenterparams.xml file. If you set this value to zero, you do not need to use the SLkn, SStn, and STyn parameters or the SearchAttr, SearchOperation, and SearchString parameters, and all members of the group are returned in the search.

This function gets all or some members of the specified group. The search is allowed only on one field. You can only have only one triplet of STy1, SLk1 and SSt1 or SearchString, SearchOperation, and SearchAttr. If you don't specify any search parameters, this function returns all members of the group if the administrator has set the minimum search length to be 0 through the groupMemberSearchStringMiminumLength parameter in the *COREid Server*/oblix/apps/groupservcenter/bin/groupservcenterparams.xml file. If this is set to 0, then the search returns all the members of the group. If this is not 0, then the search triplet must be specified, plus the string to search for (as specified thru SSt1) should have at least the same number of characters as specified by the groupMemberSearchStringMiminumLength parameter.

In order to view group members, the access control requirements are:
1. To view any members (Static, Dynamic, Nested), you need to have View right on the Member attribute;
2. To view Dynamic members, you also need to have View right on the "Dynamic Filter" attribute;

Rights: To view any members (Static, Dynamic, Nested), you need to have the View right on the Member attribute. To view. Dynamic members, you must additionally have the View right on the Dynamic Filter attribute.

WSDL file: *WebPass_install_dir*\oblix\WebServices\WSDL\gm_**viewGroupMembers**.wsdl

Parameters:

`uid` The DN of the group whose members are to be listed.

Rules: Required. Single value, a DN.

`attrName` Optional. Use one or more instances of this to specify the attributes that you want to see for each group member. Use the LDAP name of each attribute, not its display name. If you do not specify particular attributes the default is to show the class attribute of the person object class. See "Pre-NetPoint 6.5 Attribute Parameters" on page 145 or "NetPoint 6.5 and Higher Attribute Parameters" on page 149.

`memberIDsOnly` When this flag is set to true, attributes requested with the attrName parameter are ignored, with the exception of the class attribute and attributes matching any search criteria.

Rules: Optional. Boolean, true or false.

Default: false.

`showDynamicUserMembers`
Specifies whether dynamic members of a group are to be included in the response.

Rules: Optional. At least one of the show parameters in the list must be used and be set to true. Single value, Boolean, true or false.

Default: false.

`showNestedUserMembers`
Specifies whether nested members of a group are to be included in the response.

Rules: Optional. Single value, Boolean, true or false.

Default: false

`showStaticUser Members`—Optional.

`SLkn` Required (old syntax). See “Pre-NetPoint 6.5 Search Parameters” on page 140.

`SStn` Required (old syntax). See “Pre-NetPoint 6.5 Search Parameters” on page 140.

`STyn` Required (old syntax). See “Pre-NetPoint 6.5 Search Parameters” on page 140.

`SearchAttr` Required (new syntax). See “NetPoint 6.5 Search Parameters” on page 142.

`SearchOperation` Required (new syntax). See “NetPoint 6.5 Search Parameters” on page 142.

`SearchString` Required (new syntax). See “NetPoint 6.5 Search Parameters” on page 142.

`memberIDsOnly` Optional. Values are true or false. If you set this flag, only the class attribute is returned even if other attributes are requested. This limitation exists because NetPoint reads the data from the cache rather than the directory. This feature only takes affect after the first request, after all caches are initialized. The advantage of this flag is that directory hits are minimized.

Expand group

Function: expandGroup

Request example:
```
<oblix:request application="groupservcenter"
function="expandGroup">
```

Description: Expands a dynamic group into its current static members.

Rights: To expand a group, the user must have the view rights for the group name and for the attributes Group Dynamic Filter and Group Expansion, and modify rights for the Member attribute. Group Dynamic Filter is the attribute that is of semantic type, dynamic filter. The Group Expansion attribute is the attribute of type, obgroupexpandeddynamic in the oblixadvancedgroup auxiliary objectclass. Member is the attribute that is of semantic type, Static Member.

WSDL file: *WebPass_install_dir*\oblix\WebServices\WSDL\gm_**expandGroup**.wsdl

Parameters:

`groupsToExpand` A target group you want to expand. One or the other of these must be provided.

Rules: Optional. Multivalued, a DN.

`expandAllGroups` Expands all groups that you have rights to expand. If set to true, then all such groups are expanded. If set to false, then only the groups specified with the groupsToExpand parameter are expanded.

Rules: Optional. Single value, Boolean, true or false.

Default: false

Flush the Group Cache

Function: flushGroupCache

Request example:
```
<oblix:request application="groupservcenter"
function="flushGroupCache">
```

Description: Use this function to remove groups from the group cache. One of the two parameters shown below must be provided.

Rights: To flush the group cache, the user must be a NetPoint Administrator.

WSDL file: *WebPass_install_dir*\oblix\WebServices\WSDL\gm_**flushGroupCache**.wsdl

Parameters:

`flushGroup` Optional. Removes from the group cache those groups whose dn is specified as a value.

`flushGroupAll` Optional.Removes all groups from the group cache if value = true.

Subscribe a user to a group

Function: subscribeUserToGroup

Request example:
```
<oblix:request application="groupservcenter"
function="subscribeUserToGroup">
```

Description: Add (subscribe) a user other than yourself to a group. The other user does not need to be logged in. The response returns the profile for the group.

Results*:* The output is the profile of the group, defined by the schema file oblix\WebServices\XMLSchema\gsc_groupprofile.xsd.

WSDL file: *WebPass_install_dir*\oblix\WebServices\WSDL\gm_**subscribeUserToGroup**.wsdl

Parameters:

`uid` The DN of the group being subscribed to. This DN must fall under the searchbases of the logged-in user.

Rules: Required. Single value, a DN.

`proxysourceuid` The DN for a non-logged-in user who is subscribing to the group. This DN must fall under the searchbases of the logged-in user

Rules: Required. Single value, a DN.

Organization Manager Functions

If you are an administrator, the Organization Manager enables you to create and delete organizations and other objects (such as floor plans and assets) that do not belong in the User Manager or Group Manager. The Organization Manager enables end users to view organizational entities such as floor plans. The organizational entities that a person can view depend upon the rights granted by a NetPoint administrator.

The following IdentityXML functions allow you to programmatically access the Organization Manager. Note that all functions follow a similar syntax:

<oblix:request application="objservcenter" function="*name*">

For example:

<oblix:request application="objservcenter" function="canIViewObjectProfile">

Functions to Test For Attribute Permissions

The following functions provide a yes or no response as to whether you or another user have read, write, delegate, and notify permissions set for a particular attribute.

Can I view an object's profile

Function: canIViewObjectProfile

Request example:
```
<oblix:request application="objservcenter"
function="canIViewObjectProfile">
```

Description: Verifies that you can view an organization's profile.

WSDL file: *WebPass_install_dir*\oblix\WebServices\WSDL\om_**canIViewObjectProfile**.wsdl

Parameters:

`uid` The DN of the organization whose profile you want to view.

Rules: Required. Single value, a DN.

Can I view an attribute in the object's profile

Function: canIViewObjectProfileAttr

Request example:
```
<oblix:request application="objservcenter"
function="canIViewObjectProfileAttr">
```

Description: Verifies that you can view a particular attribute in an organization's profile.

WSDL file: *WebPass_install_dir*\oblix\WebServices\WSDL\om_**canIViewObjectProfileAttr**.wsdl

Parameters:

`uid` The DN of the organization whose attribute you want to view.

Rules: Required. Single value, a DN.

`targetAttribute` The schema name (not the display name) for the desired attribute.

Rules: Required. Single value, a string.

Can I modify an attribute in an object's profile

Function: canIModifyObjectProfileAttr

Request example:
```
<oblix:request application="objservcenter"
function="canIModifyObjectProfile">
```

Description: Verifies that you can change a particular attribute in an organization's profile.

WSDL file: *WebPass_install_dir*\oblix\WebServices\WSDL\om_**canIModifyObjectProfileAttr**.wsdl

Parameters:

`uid` The DN of the organization whose attribute you want to change.

Rules: Required. Single value, a DN.

`targetAttribute` The schema name (not the display name) for the desired attribute.

Rules: Required. Single value, a string.

Can I request modification through a workflow of an attribute in an object's profile

Function: canIRequestObjectAttrModification

Request example:
```
<oblix:request application="objservcenter"
function="canIRequestObjectAttrModification">
```

Description: Verifies that you can change a particular attribute in an organization's profile, using a workflow.

WSDL file: *WebPass_install_dir*\oblix\WebServices\WSDL\om_**canIRequestObjectAttrModification**.wsdl

Parameters:

`uid` The DN of the organization whose attribute you want to change.

Rules: Required. Single value, a DN.

`targetAttribute` The schema name (not the display name) for the desired attribute.

Rules: Required. Single value, a string.

Can I create a new object

Function: canICreateObject

Request example:
```
<oblix:request application="objservcenter"
function="canICreateObject">
```

Description: Verifies that you can create a new organization.

WSDL file: *WebPass_install_dir*\oblix\WebServices\WSDL\om_**canICreateObject**.wsdl

Parameters:

`ObDomainName` — A subtree within which a test is being requested.

Rules: Optional. Single value, a DN.

Default: if no value is provided, NetPoint checks to see if you have the tested rights in *any* domain.

Can I delete an existing object

Function: canIDeleteObject

Request example:
```
<oblix:request application="objservcenter"
function="canIDeleteObject">
```

Description: Verifies that you can delete an existing object.

WSDL file: *WebPass_install_dir*\oblix\WebServices\WSDL\om_**canIDeleteObject**.wsdl

Parameters:

`uid` — Rules: Required. Single value, a DN.

Can this user view an object's profile

Function: canUserViewObjectProfile

Request example:
```
<oblix:request application="objservcenter"
function="canUserViewObjectProfile">
```

Description: Verifies that a non-logged in user can view an organization's profile.

WSDL file: *WebPass_install_dir*\oblix\WebServices\WSDL\om_**canUserViewObjectProfile**.wsdl

Parameters:

`uid` — The DN of the organization whose profile you want to view.

Rules: Required. Single value, a DN.

`proxysourceuid` The DN for a non-logged-in user (proxy user) whose access rights are being tested.

Rules: Required. Single value, a DN.

Can this user view an attribute in an object's profile

Function: canUserViewObjectProfileAttr

Request example:
```
<oblix:request application="objservcenter"
function="canUserViewObjectProfileAttr">
```

Description: Verifies that a non-logged in user can view a particular attribute in an organization's profile.

WSDL file: *WebPass_install_dir*\oblix\WebServices\WSDL\om_**canUserViewObjectProfileAttr**.wsdl

Parameters:

`uid` The DN of the organization whose attribute you want to view.

Rules: Required. Single value, a DN.

`proxysourceuid` The DN for a non-logged-in user (proxy user) whose access rights are being tested.

Rules: Required. Single value, a DN.

`targetAttribute` The schema name (not the display name) for the attribute.

Rules: Required. Single value, a string.

Can a user modify an attribute in an object's profile

Function: canUserModifyObjectProfileAttr

Request example:
```
<oblix:request application="objservcenter"
function="canUserModifyObjectProfileAttr">
```

Description: Verifies that a non-logged in user can change a particular attribute in an organization's profile.

WSDL file: *WebPass_install_dir*\oblix\WebServices\WSDL\om_**canUserModifyObjectProfileAttr**.wsdl

Parameters:

`uid` The DN of the object whose attribute you want to modify.

Rules: Required. Single value, a DN.

`proxysourceuid` The DN for a non-logged-in user (proxy user) whose access rights are being tested.

Rules: Required. Single value, a DN.

`targetAttribute` The schema name (not the display name) for the desired attribute.

Rules: Required. Single value, a string.

Can a user create a new object

Function: canUserCreateObject

Request example:
```
<oblix:request application="objservcenter"
function="canUserCreateObject">
```

Description: Verifies that a non-logged in user can create a new object.

WSDL file: *WebPass_install_dir*\oblix\WebServices\WSDL\om_**canUserCreateObject**.wsdl

Parameters:

`proxysourceuid` The DN for a non-logged-in user (proxy user) whose access rights are being tested.

Rules: Required. Single value, a DN.

`ObDomainName` A subtree within which a test is being requested.

Rules: Optional. Single value, a DN.

Default: if no value is provided, NetPoint checks to see if you have the tested rights in *any* domain.

Can a user delete an existing object

Function: canUserDeleteObject

Request example:
```
<oblix:request application="objservcenter"
function="canUserDeleteObject">
```

Description: Verifies that a non-logged in user can delete an existing organization.

WSDL file: *WebPass_install_dir*\oblix\WebServices\WSDL\om_**canUserDeleteObject**.wsdl

Parameters:

`proxysourceuid` The DN for a non-logged-in user (proxy user) whose access rights are being tested.

Rules: Required. Single value, a DN.

`uid` Rules: Required. Single value, a DN.

Can this user request an object attribute modification

Function: canUserRequestObjectAttrModification

Request example:
```
<oblix:request application="objservcenter"
function="canUserRequestObjectAttrModification">
```

Description: Verifies that a non-logged in user can request a change to a particular attribute in an object profile using a workflow.

WSDL file: *WebPass_install_dir*\oblix\WebServices\WSDL\om_**canUserRequestObjectAttrModification**.wsdl

Parameters:

`uid` The DN of the object whose attribute you want to modify.

Rules: Required. Single value, a DN.

`proxysourceuid` The DN for a non-logged-in user (proxy user) whose access rights are being tested.

Rules: Required. Single value, a DN.

`targetAttribute` The schema name (not the display name) for the desired attribute.

Rules: Required. Single value, a string.

Functions to Perform Organization Manager Actions

The following functions allow you to perform actions in the Organization Manager. These are get and set functions.

View object attributes

See “View user attributes” on page 93.

Modify object attributes

Function: modifyObject

Request example:
```
<oblix:request application="objservcenter"
function="modifyObject">
```

Description: Use this function to modify object attributes.

WSDL file: *WebPass_install_dir*\oblix\WebServices\WSDL\om_**modifyObject**.wsdl

Parameters:

`uid` The DN of the object whose attributes are to be changed.

Rules: Required. Single value, a DN.

`attrName` Optional. If no attrNames are specified, all the attributes of the entry that the caller has access to view are returned. This parameter is useful when you want to modify a few attributes and only want those attributes back in the result of a modify call. This can save on performance when the profile contains a large number of attributes. See “Pre-NetPoint 6.5 Attribute Parameters” on page 145 or “NetPoint 6.5 and Higher Attribute Parameters” on page 149.

`attrName_n`	Required (old syntax). See “Pre-NetPoint 6.5 Attribute Parameters” on page 145 or “NetPoint 6.5 and Higher Attribute Parameters” on page 149.
`attrOperation_n`	Required (old syntax). See “Pre-NetPoint 6.5 Attribute Parameters” on page 145.
`attrOperation`	Required (new syntax). See “NetPoint 6.5 and Higher Attribute Parameters” on page 149.
`attrValue_n`	Required (old syntax). See “Pre-NetPoint 6.5 Attribute Parameters” on page 145.
`attrNewValue`	Required (new syntax). Required. See “NetPoint 6.5 and Higher Attribute Parameters” on page 149.
`NoOfFields`	Required. Required. See “Pre-NetPoint 6.5 Attribute Parameters” on page 145 or “NetPoint 6.5 and Higher Attribute Parameters” on page 149.
`attrOldValue`	Required/Optional. Required only if the attrOperation is a REPLACE.

Request object attribute change through a workflow

See “Request user attribute change through a workflow” on page 96.

Create an object

See “Create User” on page 99.

Self-registration

See “Self-Registration” on page 102.

Delete object

Function: delete (organization)

Request example:
```
<oblix:request application="objservcenter"
function="delete">
```

Description: Use this function to delete an organization.

WSDL file: *WebPass_install_dir*\oblix\WebServices\WSDL\om_**delete**.wsdl

Parameters:

`uid`	The DN of the group or whose attributes are to be changed. Rules: Required. Single value, a DN.
`ObWorkflowName`	The name of the workflow that you want to use to create or change the value(s) for an attribute. Find the full DN for ObWorkflowName under the view menu for workflow definition under the particular application. Rules: Required. Single value, a DN.
`ObWfComment`	Provides a comment for a step in a workflow. Rules: Optional. Single value, string.
`NoOfFields`	Optional. Required. See “Pre-NetPoint 6.5 Attribute Parameters” on page 145 or “NetPoint 6.5 and Higher Attribute Parameters” on page 149.
`attrName_n`	Optional (old syntax). The attribute specified should be an attribute configured in the COREid System console, and it should be part of one of the panels configured for the View Profile of the user, group, or organization. Otherwise, it is considered invalid. See “Pre-NetPoint 6.5 Attribute Parameters” on page 145.
`attrName`	Optional (new syntax). The attribute specified should be an attribute configured in the COREid System console, and it should be part of one of the panels configured for the View Profile of the user, group, or organization. Otherwise, it is considered invalid. See “NetPoint 6.5 and Higher Attribute Parameters” on page 149.
`attrValue_n`	See “Pre-NetPoint 6.5 Attribute Parameters” on page 145.
`attrValue`	See “Pre-NetPoint 6.5 Attribute Parameters” on page 145 or “NetPoint 6.5 and Higher Attribute Parameters” on page 149.
`attrNewValue`	Required (new syntax). Required. See “NetPoint 6.5 and Higher Attribute Parameters” on page 149.

`attrOldValue_n` Optional/Required (old syntax). Required only if the attrOperation is a REPLACE. See “Pre-NetPoint 6.5 Attribute Parameters” on page 145.

`attrOldValue` Optional/Required (new syntax). Required only if the attrOperation is a REPLACE.

Search Parameters

The following are parameters that you use with functions that conduct search operations. This section describes parameters used pre-NetPoint 6.5 and parameters for NetPoint 6.5 and higher.

Pre-NetPoint 6.5 Search Parameters

Search parameters prior to NetPoint 6.5 took an argument of n to indicate a tuple, that is, a group of related parameters. For example:

```
<oblix:params>
   <oblix:param name="tab_id">Employees</tab_id>
   <oblix:param name="STy1">cn</oblix:param>
   <oblix:param name="SLk1">OSM</oblix:param>
   <oblix:param name="SSt1">john</oblix:param>
</oblix:params>
```

In this example, the number 1 differentiates this search tuple from any other search tuples in the XML file.

The following are pre-NetPoint 6.5 search parameters.

Parameter: STyn

Description: An attribute whose string values are to be searched. Attributes are associated, by application, with one or more tabs. The attribute must have been marked as searchable for the tab name provided by the tab_id parameter.

An administrator must set the searchable flag for the attribute.

Rules: Required. Multivalued, 1 to *n*. For an explanation of *n*, see the noOfFields parameter.

Parameter: SLkn

Description:	The way string data is to be selected. Legal entries all begin with the letter O, and the next two letters are an abbreviation of the search type. Possible values are: **OSM**—Substring match. Search results include entries whose value contains the data entered for this parameter. **OGE**—Greater than or equal to. Search results include entries whose string value is greater than or equal to the data entered for this parameter. **OLE**—Less than or equal to. Search results include entries whose string value is greater than or equal to the data entered for this parameter. **OBW**—Begins with. Search results include entries whose string value begins with the data entered for this parameter. **OEW**—Ends with. Search results include entries whose string value ends with the data entered for this parameter. **OSL**—Sounds like. Attempts a phonic match on the entered data. **OEM**—Exact match. Search results include entries whose string value is the same as the data entered for this parameter. **OOS**—Oblix-specific substring match. Differs from OSM. Multiple search strngs can be entered, delimited by spaces. Results include entries that match both of the two strings. Any other value than the ones specified above returns an error (Invalid parameters).
Rules:	Required. Multivalued, 1 to *n*. For an explanation of *n*, see noOfFields.
Default:	None. If an invalid value or no value is provided, an error is returned.
Parameter:	SStn
Description:	Use this parameter to provide a string value to be searched for.

Rules:	Required. Multivalued, 1 to *n*. For an explanation of *n*, see the noOfFields parameter.
Default:	If no value is specified, then the default is to do a blank search on the class attribute. This means, return everything that has any value (but not a NULL value) for the selected STy attribute.
Parameter:	**noOfFields** (when used with search)
Description:	The number of attributes whose values are to be searched. Depending on the value of this parameter, you must provide the same number of sets of STy, SLk and SSt parameters. For example, if the noOfFields is 2, required parameters would be STy1, SLk1 and SSt1 and STy2, SLk2 and SSt2. The result of the search is an AND that satisfies all of the parameter sets. The value for noOfFields must be greater than or equal to the number of sets. If it is greater, the behavior is as if you had entered the correct value for *n*.
Rules:	Optional. Single value, an integer value >= 1.
Default:	1

NetPoint 6.5 Search Parameters

In NetPoint 6.5 and higher versions, search parameters are expressed using a condition tag to delimit a tuple. The parameter names have changed to be more congruent with the operation being performed. An example:

```
<oblix:tab_id>Employees</oblix:tab_id>
<oblix:SearchParams>

   <oblix:noOfFields>2</oblix:noOfFields>
   <oblix:Condition>
        <oblix:SearchAttr>cn</oblix:SearchAttr>
        <oblix:SearchOperation>OSM</oblix:SearchOperation>
        <oblix:SearchString>john</oblix:SearchString>
   </oblix:Condition>
   <oblix:Condition>
        <oblix:SearchAttr>cn</oblix:SearchAttr>
        <oblix:SearchOperation>OSM</oblix:SearchOperation>
        <oblix:SearchString>mary</oblix:SearchString>
   </oblix:Condition>
</oblix:SearchParams>
```

The NetPoint 6.5 search parameters are as follows.

Parameter: searchAttr

Description: An attribute whose string values are to be searched. Attributes are associated, by application, with one or more tabs. The attribute must have been marked as searchable for the tab name provided by the tab_id parameter. If it is not, an error is returned.

An administrator must have set the searchable flag for the attribute.

Rules: Required. Multivalued, 1 to *n*. For an explanation of *n*, see the noOfFields parameter.

Parameter: searchOperation

Description: The way string data is to be selected. Legal entries all begin with the letter O, and the next two letters are an abbreviation of the search type.

Possible values are:

OSM—Substring match. Search results include entries whose value contains the data entered for this parameter.

OGE—Greater than or equal to. Search results include entries whose string value is greater than or equal to the data entered for this parameter.

OLE—Less than or equal to. Search results include entries whose string value is greater than or equal to the data entered for this parameter.

OBW—Begins with. Search results include entries whose string value begins with the data entered for this parameter.

OEW—Ends with. Search results include entries whose string value ends with the data entered for this parameter.

OSL—Sounds like. Attempts a phonic match on the entered data.

OEM—Exact match. Search results include entries whose string value is the same as the data entered for this parameter.

OOS—Oblix-specific substring match. Differs from OSM. Multiple search strngs can be entered, delimited by spaces. Results include entries that match both of the two strings.

Any other value than the ones specified above returns an error (Invalid parameters).

Rules: Required. Single value.

Default: None. If an invalid value or no value is provided, an error is returned.

Parameter: SearchString

Description: Use this parameter to provide a string value to be searched for.

Rules: Required. Single value.

Default: If no value is specified, then the default is to do a blank search on the class attribute. This means, return everything that has any value (but not a NULL value) for the selected searchAttr attribute.

Parameter: **noOfFields** (when used with search)

Description: The number of attributes whose values are to be searched. Depending on the value of this parameter, you must provide the same number of conditions of SearchAttribute, SearchOperation and SearchString parameters. For example, if the noOfFields is 2, you would need to supply two conditions in the SearchParams element, and specify a set of search parameters within each condition.

The result of the search is an AND that satisfies all of the parameter sets.

The entered or default value for noOfFields must be greater than or equal to the number of sets. If it is greater, no error is reported, and the behavior is as if you had entered the correct value for *n*.

Rules: Optional. Single value, an integer value >= 1.

Default: 1

Attribute Parameters

The following parameters are used for operations on attributes.

Pre-NetPoint 6.5 Attribute Parameters

Prior to NetPoint 6.5, attribute parameters were specified using *n* to indicate a tuple. For example:

```
<oblix:params>
<oblix:param name="attrName_1">userPassword</oblix:param>
<oblix:param name="attrValue_1">password</oblix:param>
<oblix:param name="attrValue_1_confirm">password</oblix:param>
<oblix:param name="attrOldValue_1">d</oblix:param>
<oblix:param name="attrOperation_1">REPLACE</oblix:param>
<oblix:param name="noOfFields">1</oblix:param>
</oblix:params>
```

The following are attribute parameters.

Parameter: attrName_n

Description: The LDAP names of one or more attributes to be viewed or changed. Use the schema names from the directory, not the display names. The attribute must be one of the configured attributes in NetPoint and should be part of one of the panels configured for that user, group, or object profile.

A given attrName_n combination must appear only once. If it appears more than once, every operation except the first on that attribute name is ignored. Invalid attribute names or attribute names that are not shown as part of a panel for the given function, are ignored.

Optional for functions such as view or myGroupsProfile. In these cases, the _n extension is not used.

Rules: Required/Optional. Multivalued, string, 1 to *n*. For an explanation of *n*, see noOfFields.

Default: If no names are provided, only the attributes that the user is allowed to view are considered, depending upon the function.

Parameter: attrOldValue_n_m_suffix

Description:	Use this parameter for changes, to specify the old value for the attribute named by attrName_n. The rules for the use of m and suffix are the same as for attrValue.
Rules:	Required if the attrOperation is a replace, otherwise ignored. Multivalued, string, 1 to *n*. For an explanation of *n*, see noOfFields.
Parameter:	attrOperation_n
Description:	The type of operation to perform on the attribute. Legal values are: **ADD**—Add the attribute name and value to the existing attributes. You will get an error if the combination exists already. **DELETE**—Delete the attribute and value combination from the existing attributes. You will get an error if the combination does not exist. **REPLACE**—Delete the old attribute name and value combination and replace it with the new attribute name and value combination. If you use REPLACE, you *must* also use the attrOldValue_n parameter. **REPLACE_ALL**—Delete the old attribute and name combinations and replace them with new attribute name and value combinations. Use this if you do not care what the old values were, and just want to replace all of them. In this case, attrOldValue_n is not used.Any other value returns an error message such as "Invalid value for attibute abcd". Operations on attributes of display type “location”, for example obparent locationdn, are not supported through IdentityXML.
Rules:	Required. Multivalued, string, 1 to *n*. For an explanation of *n*, see noOfFields.
Parameter:	attrValue_n_m_suffix
Description:	The value that you want to add, delete, or replace for the attribute specified with attrname. See the Rules column for an explanation of m.

Some attributes can have more than one value. To provide these, use attrValue_n again, using the same n.

The trailing m and suffix are usually not needed. However, values for some attributes are defined using multiple fields, whose content must be defined using suffixes. These attributes are:

Date attribute—Dates contain at least three fields which must be specified, the day, month and year. Content must match the syntax defined for DATETYPE in globalparams.xml. Specify these using the suffixes _day, _month and _year.

DATETYPE ObIntegerDate takes additional fields, for hours, minutes and seconds, which use the suffixes _hours, _minutes, and _seconds, respectively.

DATETYPE ObISO8061Date takes at least one additional field, _bahead_utc, to allow for a time zone offset. If the content of this suffix is Z, there is no offset and no addtional suffixes. If the content is + or - then the offset must be supplied in hours and minutes, using the suffixes _tz_hours and _tz-minutes, respectively.

Password attribute—Passwords contain three fields which must be specified: the new value, confirm value and old value. New value does not take a suffix. The confirm value and the old value use suffixes _confirm and _old, respectively.

For an add or modify operation: you must specify the new password as well as theconfirmation (as attrValue_3 and attrValue_3_confirm). If you modify your own password, you need to also provide the old value of the password, in attrValue_3_old.

The attrOperation in this case can be any of the values ADD, REPLACE, or REPLACE_ALL.

If you need to delete the password-type attribute, specify the attrOperation as DELETE. You do not need to provide the old value or new value parameters.

Postal address attribute—Postal addresses allow for six fields. Specify each field by name: _field_0, _field_1, and so on.

Legal values for attributes generally match what is shown in the GUI that corresponds to the function to be executed. Some exceptions exist, however, and are described at "Exceptions to Attribute Values" on page 154.

Rules: Required. Multivalued string, 1 to *n*. For an explanation of *n*, see noOfFields.

Default: none. If an attribute takes multiple values and the values are made up of multiple fields (therefore requiring suffixes to be used), then use m to define which value the suffix data applies to.

Example: This is an example for the use of m, as mentioned in the discussion of the attrvalue attribute. In this case, the attribute named AnniversaryDates is the fourth attribute being modified; the n value is therefore 4. It is a date type attribute, which takes six fields and suffix combnations to fully describe its value. Two values are added for it; the m values are 1 and 2.

Listing 24 Attribute Name Example (old syntax)

```
<oblix:param name="noOfFields">4</oblix:param>
....
....
<oblix:param name="attrName_4">AnniversaryDates</oblix:param>
<oblix:param name="attrValue_4_1_day">12</oblix:param>
<oblix:param name="attrValue_4_1_month">6</oblix:param>
<oblix:param name="attrValue_4_1_year">1996</oblix:param>
<oblix:param name="attrValue_4_1_hours">10</oblix:param>
<oblix:param name="attrValue_4_1_minutes">45</oblix:param>
<oblix:param name="attrValue_4_1_seconds">55</oblix:param>
<oblix:param name="attrValue_4_2_day">12</oblix:param>
<oblix:param name="attrValue_4_2_month">6</oblix:param>
<oblix:param name="attrValue_4_2_year">2006</oblix:param>
<oblix:param name="attrValue_4_2_hours">10</oblix:param>
<oblix:param name="attrValue_4_2_minutes">45</oblix:param>
<oblix:param name="attrValue_4_2_seconds">55</oblix:param>
<oblix:param name="attrOperation_4">ADD</oblix:param>
```

Parameter: **noOfFields** (when used with workflow/modify attributes)

Description: The number of attributes you want to modify. You attach a unique value *n* to each of the parameters attrName, attrValue, attrOperation, and (sometimes) attrOldValue, that ties them together, telling the application what operation and what values are to be applied (and sometimes changed) for each attribute.

You must then provide *n* sets of attrName, attrValue, attrOperation, and (sometimes) attrOldValue parameters. For example, if noOfFields is 2, NetPoint expects additional parameters.

attrName_1, attrValue_1, attrOperation_1, and attrOldValue_1 and attrName_2, attrValue_2, attrOperation_2, and attrOldValue_2.

Rules: Required. Single value, integer.

NetPoint 6.5 and Higher Attribute Parameters

In NetPoint 6.5 and higher, attribute parameters are expressed as follows:

```
<oblix:AttributeParams>
   <oblix:GenericAttribute>
   <!--Generic string type attribute-->
      <oblix:AttrName>genphonenumber</oblix:AttrName>
      <oblix:AttrOperation>REPLACE</oblix:AttrOperation>
      <oblix:AttrOldValue>408</oblix:AttrOldValue>
      <oblix:AttrNewValue>650</oblix:AttrNewValue>
   </oblix:GenericAttribute>
</oblix:AttributeParams>
```

For each display type, there are templates for the specifying attribute values for the following: generic, password, Date, DateISO8601, and postal address. Examples:

```
<oblix:noOfFields>5</oblix:noOfFields>
<oblix:AttributeParams>

   <oblix:GenericAttribute>
   <!--Generic string type attribute-->
      <oblix:noOfFields>1</oblix:noOfFields>
      <oblix:AttrName>cn</oblix:AttrName>
      <oblix:AttrOperation>REPLACE</oblix:AttrOperation>
      <oblix:AttrOldValue>jim</oblix:AttrOldValue>
      <oblix:AttrNewValue>james</oblix:AttrNewValue>
   </oblix:GenericAttribute>

   <oblix:PasswordAttribute>
   <!--Password type attribute-->
      <oblix:AttrName>pwd</oblix:AttrName>
      <oblix:AttrOperation>REPLACE</oblix:AttrOperation>
      <oblix:AttrOldValue>mypassword</oblix:AttrOldValue>
      <oblix:AttrNewValue>mynewpassword</oblix:AttrNewValue>
   </oblix:PasswordAttribute>

   <oblix:DateAttribute>
   <!--Generic datetype attribute-->
      <oblix:AttrName>date</oblix:AttrName>
```

```
      <oblix:AttrOperation>REPLACE</oblix:AttrOperation>
      <oblix:AttrOldValue>
         <oblix:day>21</oblix:day>
         <oblix:month>7</oblix:month>
         <oblix:year>2003</oblix:year>
         <oblix:hours>22</oblix:hours>
         <oblix:minutes>33</oblix:minutes>
         <oblix:seconds>11</oblix:seconds>
      </oblix:AttrOldValue>
      <oblix:AttrNewValue>
         <oblix:day>2</oblix:day>
         <oblix:month>10</oblix:month>
         <oblix:years>2004</oblix:year>
         <oblix:hours>15</oblix:hours>
         <oblix:minutes>10</oblix:minutes>
         <oblix:seconds>3</oblix:seconds>
      </oblix:AttrNewValue>
   </oblix:DateAttribute>

   <oblix:DateAttributeISO8601>
   <!--ISO8601 date type attribute-->
      <oblix:AttrName>date</oblix:AttrName>
      <oblix:AttrOperation>REPLACE</oblix:AttrOperation>
      <oblix:AttrOldValue>
         <oblix:bahead_utc>2100</oblix:bahead_utc>
         <oblix:tz_hours>22</oblix:tz_hours>
         <oblix:tz_minutes>33</oblix:tz_minutes>
      </oblix:AttrOldValue>
      <oblix:AttrNewValue>
         <oblix:bahead_utc>400</oblix:bahead_utc>
         <oblix:tz_hours>10</oblix:tz_hours>
         <oblix:tz_minutes>8</oblix:tz_minutes>
      </oblix:AttrNewValue>
   </oblix:DateAttributeISO8601>

   <oblix:PostalAddressAttribute>
   <!--Postal address type attribute-->
      <oblix:AttrName>addr</oblix:AttrName>
      <oblix:AttrOperation>REPLACE</oblix:AttrOperation>
      <oblix:AttrOldValue>
         <oblix:field>123 Main St.</oblix:field>
         <oblix:field>San Jose</oblix:field>
      </oblix:AttrOldValue>
      <oblix:AttrNewValue>
      <oblix:noOfFields>2</oblix:noOfFields>
         <oblix:field>100 Forge Dr.</oblix:field>
         <oblix:field>Cupertino</oblix:field>
      </oblix:AttrNewValue>
   </oblix:PostalAddressAttribute>
</oblix:AttributeParams>
```

Note that in the NetPoint 6.5 syntax, the noOfFields parameter is specified outside of the <oblix:AttributeParams> tag. The noOfFields parameter refers to the total number of attributes being specified. Each attribute must be enclosed in the appropriate tag element delimiters (for PostalAddress, GenericAttribute, and so on). An example:

```
<oblix:noOfFields>3</oblix:noOfFields>
<oblix:AttributeParams>
   <oblix:GenericAttribute>
        <oblix:AttrName>cn</oblix:AttrName>
        <oblix:AttrOperation>REPLACE</oblix:AttrOperation>
        <oblix:AttrOldValue>jim</oblix:AttrOldValue>
        <oblix:AttrNewValue>james</oblix:AttrNewValue>
   </oblix:GenericAttribute>
   <oblix:GenericAttribute>
        <oblix:AttrName>title</oblix:AttrName>
        <oblix:AttrOperation>REPLACE</oblix:AttrOperation>
        <oblix:AttrOldValue>development</oblix:AttrOldValue>
        <oblix:AttrNewValue>sales</oblix:AttrNewValue>
   </oblix:GenericAttribute>
   <oblix:PostalAddressAttribute>
        <oblix:AttrName>addr</oblix:AttrName>
        <oblix:AttrOperation>REPLACE</oblix:AttrOperation>
        <oblix:AttrOldValue>
        <oblix:field>123 Main St.</oblix:field>
        <oblix:field>San Jose</oblix:field>
        </oblix:AttrOldValue>
        <oblix:AttrNewValue>
        <oblix:field>100 Forge Dr.</oblix:field>
        <oblix:field>Cupertino</oblix:field>
        </oblix:AttrNewValue>
   </oblix:PostalAddressAttribute>
</oblix:AttributeParams>
```

Parameter: attrName

Description: The names of one or more attributes to be viewed or changed. Use the schema names from the directory, not the display names. The attribute must be one of the configured attributes in NetPoint and should be part of one of the panels configured for that user, group, or object profile.

A given attrName combination must appear only once. If it appears more than once, every operation except the first on that attribute name is ignored. Invalid attribute names or attribute names that are not shown as part of a panel for the given function, are ignored.

Optional for functions such as view or myGroupsProfile.

Rules:	Required/Optional. Single value string, 1 to *n*.
Default:	If no names are provided, only the attributes that the user is allowed to view are considered, depending upon the function.
Parameter:	attrOldValue
Description:	Use this parameter for changes, to specify the old value for the attribute named by attrName.
Rules:	Required if the attrOperation is a replace, otherwise ignored. Single value string. Multiple instances of attrOldValue can be replaced by the value supplied in attrNewValue.
Parameter:	attrOperation
Description:	The type of operation to perform on the attribute. Legal values are: **ADD**—Add the attribute name and value to the existing attributes. You will get an error if the combination exists already. Valid for LDAP attributes only. **DELETE**—Delete the attribute and value combination from the existing attributes. You will get an error if the combination does not exist. **REPLACE**—Delete the old attribute name and value combination and replace it with the new attribute name and value combination. If you use REPLACE, you must also use the attrOldValue parameter. Valid for LDAP attributes only. **REPLACE_ALL**—Delete the old attribute and name combinations and replace them with new attribute name and value combinations. Use this if you do not care what the old values were, and just want to replace all of them. In this case, attrOldValue is not used.Any other value returns an error message such as "Invalid value for attribute". Can be specified for LDAP attributes and template attributes. Operations on attributes of display type “location”, for example obparent locationdn, are not supported through IdentityXML.
Rules:	Required. Single value string.

Parameter:	attrNewValue
Description:	The value that you want to add, delete, or replace for the attribute specified with attrName. Some attributes can have more than one value. To provide these, use attrNewValue again. Dates contain at least three fields which must be specified, the day, month and year. Content must match the syntax defined for DATETYPE in oblixbaseparams.xml. DateAttributeISO8601 takes at least one additional field, bahead_utc, to allow for a time zone offset. If the content of this suffix is Z, there is no offset and no additional suffixes. If the content is + or - then the offset must be supplied in hours and minutes, using the tz_hours and tz-minutes, respectively. Passwords contain three fields which must be specified: the new value, confirmation value, and old value. For an add or modify operation: you must specify the new password as well as the confirmation. If you modify your own password, you need to also provide the old value of the password. The attrOperation in this case can be any of the values ADD, REPLACE, or REPLACE_ALL. If you need to delete the password-type attribute, specify the attrOperation as DELETE. You do not need to provide the old value or new value parameters. Postal addresses allow for six fields. Legal values for attributes generally match what is shown in the GUI that corresponds to the function to be executed. Some exceptions exist, however, and are described at “Exceptions to Attribute Values” on page 154.
Rules:	Required. Single value
Parameter:	noOfFields (when used with workflow/modify attributes)
Description:	The number of attributes to be modified.
Rules:	Required. Single value, integer.

Exceptions to Attribute Values

In general, legal values for attributes used in the functions match those that are used in the GUI. However, exceptions exist, as described in the following table.

Table 1 Exceptional Attribute Values

Attribute Name	Description	Values
`obgroupsubscribe notification`	If this attribute is set, the affected UID will be notified when the UID is subscribed or unsubscribed from a group.	**NotifyUponSubscription**—If the user is to be notified when subscribed to a group (matches subscribe for the GUI). **NotifyUponUnsubscription**—If the user is to be notified when unsubscribed from a group (matches unsubscribe for the GUI).
`obgroup subscriptiontype`	This attribute is set to define the limits under which users can be subscrbed to the group.	**SubscriptionPolicyOpen**—Matches Open for the GUI. **SubscriptionPolicyOpenFilter**—Matches Open with Filter for the GUI. **SubscriptionPolicyControlledWorkflow** —Matches Controlled through Workflow for the GUI. **SubscriptionPolicyClosed**—Matches Closed for the GUI.

Examples

The following sections provide examples of how IdentityXML can be packaged and deployed.

Java Application Example

The example in Listing 25: "Java Example to Query NetPoint using SOAP." on page 155 sends a SOAP message contained in a file to a COREid System application using HTTP. It receives the data from the HTTP response and prints it on the console.

This program illustrates the basics of IdentityXML programming. For instance, NetPoint can be scripted using techniques like this. You can write small programs that do one thing, such as transmit a pre-composed message, and glue them together in Perl or shell scripts to perform more complex tasks with NetPoint. The intelligence concerning the message contents need not be part of the IdentityXML client.

Listing 25 Java Example to Query NetPoint using SOAP.

```
/**
 * This is a simple example of how to query Oblix NetPoint
 * using a SOAP message.
 *
 * This program sends a SOAP request contained in a file
 * file to an COREid System application (User Manager by default)
 * on a specified host:port (localhost:80 by default).
*
* Requirements:
 * HTTPClient, an HTTP client library from innovation,
 * is required to run this program.
 * HTTPClient is freely available at
 *    http://www.innovation.ch/java/HTTPClient
 * This program has been tested with HTTPClient Version 0.3-2
 *
 * To run:
 * java Lookup -f inputfile [-h hostname] [-p port] [-u oblixurl]
 *
 */

import HTTPClient.*;
import java.util.*;
import java.io.*;

public class Lookup
{
   static String hostname = "localhost";
   static String filename = null;
   static int port = 80;
   static String oburl = "/identity/oblix/apps/userservcenter/
        bin/userservcenter.cgi";

(CONTINUED)
```

Listing 25 (Continued) Java Example to Query NetPoint using SOAP.

```
   public static void collectArgs(String args[])
   {
      for (int i = 0; i < args.length; i++) {
         if (args[i].equals("-h") && args.length >= i+1)
            hostname = args[i+1];
         else if (args[i].equals("-f") && args.length >= i+1)
            filename = args[i+1];
         else if (args[i].equals("-p") && args.length >= i+1)
            port = Integer.parseInt(args[i+1]);
         else if (args[i].equals("-u") && args.length >= i+1)
            oburl = args[i+1];
         else if (args[i].equals("-h")||
               args[i].equals("-help")
            {
            System.out.println("Usage: java Lookup -f filename
                  [-h hostname] [-p port]  [-u oblixurl] \n");
         }
      }
   }
   public static void collectArgs(String args[])
   {
      for (int i = 0; i < args.length; i++) {
         if (args[i].equals("-h") && args.length >= i+1)
            hostname = args[i+1];
         else if (args[i].equals("-f") && args.length >= i+1)
            filename = args[i+1];
         else if (args[i].equals("-p") && args.length >= i+1)
            port = Integer.parseInt(args[i+1]);
         else if (args[i].equals("-u") && args.length >= i+1)
            oburl = args[i+1];
         else if (args[i].equals("-h")||
               args[i].equals("-help")
            {
            System.out.println("Usage: java Lookup -f filename
                  [-h hostname] [-p port]  [-u oblixurl] \n");
         }
      }
   }

(CONTINUED)
```

Listing 25 (Continued) Java Example to Query NetPoint using SOAP.

```
/**
* THIS is a simple example of how to query Oblix NetPoint
 * using a SOAP message.
 * This program sends a SOAP request contained in a file
 * to an COREid application (User Manager by default)
 * on a specified host:port (localhost:80 by default).
*
* Requirements:
 * HTTPClient, an HTTP client library from innovation,
 * is required to run this program.
 * HTTPClient is freely available at
 *    http://www.innovation.ch/java/HTTPClient
 * This program has been tested with HTTPClient Version 0.3-2
 *
 * To run:
 * java Lookup -f inputfile [-h hostname] [-p port] [-u oblixurl]
*/

import HTTPClient.*;
import java.util.*;
import java.io.*;

public class Lookup
{
   static String hostname = "localhost";
   static String filename = null;
   static int port = 80;
   static String oburl = "/identity/oblix/apps/userservcenter/
        bin/userservcenter.cgi";

   /**
    * Read in specified file and return as string.
    */
   public static String getRequestFromFile()
   {
      StringBuffer data = new StringBuffer();
      try {
         BufferedReader reader = new BufferedReader
               (new FileReader(filename));
         for (String line = reader.readLine(); line != null;
            line = reader.readLine()) {
            data.append(line);
         }
      } catch (Exception e) {
         System.out.println(e.toString());
      }
      return data.toString();

(CONTINUED)
```

Listing 25 (Continued) Java Example to Query NetPoint using SOAP.

```
   public static void main(String args[]) throws Exception
   {
      collectArgs(args);
      if (null == filename) {
         System.out.println("Usage: java Lookup -f filename
               [-h hostname] [-p port]  [-u oblixurl] \n");
      }
      try {

         // Set up HTTP request containing SOAP message,
         // and send the request to the server.
         CookieModule.setCookiePolicyHandler(null);
         HTTPConnection con = new HTTPConnection(hostname, port);
         NVPair header[] = new NVPair[1];
         header[0] = new NVPair("Content-Type", "text/xml");
         HTTPResponse rsp =
            con.Post(oburl,
            getRequestFromFile(),
            header);

         // Check for HTTP errors
         if (rsp.getStatusCode() >= 300) {

            System.err.println("Received Error:
                  "+rsp.getReasonLine());
            System.err.println(new String(rsp.getData()));
         } else {
            // Send the output to stdout
            System.out.println(rsp.getText());

         }
      } catch (IOException ioe) {
         System.err.println(ioe.toString());
      } catch (ModuleException me) {
         System.err.println("Error handling request:
               " + me.getMessage());
      } catch (Exception e) {
         System.out.println(e.toString());
      }
   }
}
```

The application's main() method begins by collecting command-line arguments, of which only the filename is required, the others being defaulted. It then uses the HTTPClient package to connect to the server running WebPass, and constructs an HTTP request to transport the SOAP request. It POSTs the request, extracts the response from the HTTP connection object, and sends it to standard output.

Examples of SOAP messages you can send using this program are installed in:

```
unsupported/integsvcs/*.xml
```

You should inspect the files, copy the one you wish to send into the current directory of the application, and modify the request parameters within the XML file to specify parameters that make sense for your directory (such as valid uid for a view request). Make sure you have that HTTPClient package from Innovation in your CLASSPATH. Then send the request with the following command:

```
java Lookup -f inputfile [-h hostname] [-p port]
[-u oblixurl]
```

Java Servlet Example

This example builds on the previous one. This example is a Java Servlet that runs within a Web or application server. Since simply printing out the result is not useful for a servlet, this one shows a simple example of using a NetPoint attribute value to dynamically generate an HTML page. The servlet assumes your SOAP message invokes the User Manager view program, and gets user profile data as a response. It then uses the JAXP XML parser to parse the SOAP message containing the user profile, and extracts the email address attribute for the user being viewed.

If you log into User Manager before running this servlet, and edit the email attribute for the user you are going to look up so that it reads red or green, you will see that the value is used by the servlet in generating the resulting HTML page. It uses this value for the BGCOLOR attribute of the BODY element on the page, as well as printing it out. If you specify a valid color or #RRGGBB value, the page is displayed in that color.

The servlet assumes the following request to view a user profile from User Manager. This request, and many other examples, are installed in:

```
unsupported/integsvcs/um_view.xml
```

You should copy and modify the request file to specify a valid user *uid* for your directory. Make sure to put it in the current working directory for the servlet, or specify the full pathname when reading the file. Listing 26: "Request File Example." on page 160 is an example request file.

Listing 26 Request File Example.

```
<?xml version="1.0" ?>
<SOAP-ENV:Envelope
      xmlns:SOAP-ENV="http://schemas-xmlsoap.org/soap/envelope/"
      xmlns:oblix="http://www.oblix.com">
   <SOAP-ENV:Body>
      <oblix:authentication
           xmlns:oblix="http://www.oblix.com" type="basic">
         <oblix:login>user1k1</oblix:login>
         <oblix:password>oblix</oblix:password>
      </oblix:authentication>
      <oblix:request function="view">
         <oblix:params>
            <oblix:name="uid">cn=Rohit Valiveti,ou=Sales,
                 ou=Dealer1k1,ou=Latin America,
                 ou=Ford,o=Company,c=US</oblix:name>
            <oblix:name="attrName">mail</oblix:name>
         </oblix:params>
      </oblix:request>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

This request could easily have been constructed in code, but it is stored as a file for the purposes of this example. Note the general structure of the SOAP message, and the parameter elements being requested.

Listing 27: "MyobServelet.java." on page 161 is the code for MyobServelet, which sends this request, parses the result, and generates an HTML page dynamically as the result of the request. MyobServelet is derived from HttpServlet. It uses JAXP, a SAX parser for XML freely available from Sun Microsystems, and HTTPClient, a freely available HTTP client package available from Innovation. You may choose to use such packages, or integrate with your own as needed.

Listing 27 MyobServelet.java.

```
/**
* This example illustrates how to use the Oblix NetPoint
* COREid System via the IdentityXML interface from a Java servlet.
* The example uses two freely available third-party Java packages:
*   1. HTTPClient, from Innovation
*        (http://www.innovation.ch/java/HTTPClient)
*   2. JAXP, from Sun Microsystems
*        (http://java.sun.com/xml/xml_jaxp.html)
* HTTPClient encapsulates the client side of the HTTP protocol.
* JAXP provides APIs for XML parsing and XSL processing.
*
* What This Servlet Does:
* This servlet reads a SOAP request from a text file /tmp/um_view.xml.
* It could quite easily have hard-coded the request, or built it
* from parameters, or fetched it from a database.  Reading it from
* a file just provides some simple flexibility and allows the code
* to be uncluttered with that detail.  The servlet uses HTTPClient
* to connect to the web server hosting the NetPoint COREid System,
* sends the SOAP message to the User Manager application's URL
* (hard-coded in this example), and receives the response.  It then
* creates an XML parser, and uses a custom document handler,
* MyObReader, to handle only those elements of interest during the
* parse. In this case, MyObReader only cares about the ObEmail
* element.  When it finds the element, it stores the value, which
* is then available to this servlet via the MyObReader.getEmail()
* method after parsing.
*
* As an HTTPServlet, a natural response for this servlet is an
* HTML page.  To illustrate use of NetPoint data in building the
* page returned, this servlet does something a little unusual: it
* uses the value of the user's email address as a color, and uses
* it to set BGCOLOR attribute of the BODY element in the output HTML.
*
* To demonstrate the example, you should edit the SOAP message to
* specify the DN of the user profile in your COREid System that
* you want to use.  Then, logged in as an NP admin, change the
* user's email address to a valid HTML color value. like "green"
* or an RGB value, like "#ffddff".  Then run the servlet (i.e.
* install in your WS and fetch .../servlets/EmailColor). The
* page built by the servlet should appear, with its background
* rendered in the color you saved as the email address.
*
* The helper class, MyObReader, extends the DefaultHandler content
* handler of JAXP by adding handler methods for the ObEmail element
* and its nested ObValue element containing the email address,
* which is what this example is looking up. After the parse,
* the helper class is queried for the email address, and this
* is added to the HTTPSerlvlet output stream (a simple HTML page)
* which is sent back to the browser.
```

Listing 27 (Continued) MyobServelet.java.

```
* To run:
*  Edit the SOAP message file, and make sure the hard-coded
*  path in this class points to where you saved the SOAP message.
*  Build the MyObServlet class and the MyObReader helper class.
*  Put the class files and JAR files for these two classes, plus
*  the HTTPClient package and JAXP in your web server's classpath
*  for servlets. For iPlanet Web Server, you can find this
*  in the console under
*  Servlets/Configure JVM Attributes/Classpath.
*  Restart your web server if necessary.
*  Point your browser at <yourServletRoot>/ObSoapClient
*/
import java.io.*;
import java.net.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
import HTTPClient.*;
// JAXP packages
import javax.xml.parsers.*;
import org.xml.sax.*;
import org.xml.sax.helpers.*;
public class MyObServlet extends HttpServlet{
   // Host and port of the web server for NetPoint
   static String hostname = "localhost";
   static String filename = "c:/temp/um_view.xml"; // NT PATH !
        static int port = 80;

   // URL for User Manager
   static String oburl =
        "/identity/oblix/apps/userservcenter/bin/
           userservcenter.cgi";

   public static String getRequestFromFile(HttpServlet s){
      StringBuffer data = new StringBuffer();
      try  {
              BufferedReader reader = new BufferedReader
                    (new FileReader(filename));

              for (String line = reader.readLine(); line != null;
                                  line = reader.readLine()){
                 data.append(line);
              }
           } catch (Exception e) {
                       s.getServletContext().log(e.toString());
           }
      return data.toString();
   }
```

Listing 27 (Continued) MyobServelet.java

```
public void doGet(HttpServletRequest req,
                  HttpServletResponse res)
     throws ServletException, IOException {
   try {
      CookieModule.setCookiePolicyHandler(null);
      // Initiate new HTTP connection to NetPoint's WebPass srvr
      HTTPConnection con = new HTTPConnection(hostname, port);
      // Send SOAP message, collect response
      NVPair header[] = new NVPair[1];
      header[0] = new NVPair("Content-Type", "text/xml");
      String rqString = getRequestFromFile(this);
      getServletContext().log("request is: " + rqString);
      HTTPResponse rsp = con.Post(oburl, rqString, header);

      // Check HTTP status and act accordingly
      if (rsp.getStatusCode() >= 300) {

         getServletContext().log("Received Error:
              "+rsp.getReasonLine());
         getServletContext().log(new String(rsp.getData()));

      } else {

         // HTTP success

            getServletContext().log
               ("got SOAP result. next is parsing.");

         // Create a JAXP SAXParserFactory and configure it
            SAXParserFactory spf =
                 SAXParserFactory.newInstance();
            spf.setValidating(false);

            XMLReader xmlReader = null;
            try {
            // Create a JAXP SAXParser
               SAXParser saxParser = spf.newSAXParser();

               // Get the encapsulated SAX XMLReader
               xmlReader = saxParser.getXMLReader();
            } catch (Exception ex) {
                  getServletContext().log(ex, ex.toString());
                  System.exit(1);
            }
```

Listing 27 (Continued) MyobServelet.java

```
// Set the ContentHandler of the XMLReader
              // Keep our content handler around to query later
              MyObReader myHandler = new MyObReader();
              xmlReader.setContentHandler(myHandler);
              // Set an ErrorHandler before parsing
              xmlReader.setErrorHandler
                    (new MyErrorHandler(System.err));
              try {
              // parse the XML document
                 xmlReader.parse
                       (new InputSource(rsp.getInputStream()));
                 res.setContentType("text/html");
                 PrintWriter out = res.getWriter();
                 out.println("<HTML>");
                 out.println("<HEAD>");
                 out.println("<TITLE>");
                 out.println("Oblix NetPoint via Servlet");
                 out.println("</TITLE>");
                 out.println("</HEAD>");
              // the example assumes a NetPoint admin has set
              // the email address to a color value instead
                 String emailAddress = myHandler.getEmail();
                 String bgColorAttr = "bgcolor=\"" +
                       emailAddress.trim() + "\"";
                 out.println("<BODY " + bgColorAttr + ">");
                 out.println("<H1>");
                 out.println("Oblix NetPoint via Servlet");
                 out.println("</H1>");
                 out.println("<P>");
                 out.println("The e mail address is: "
                       + emailAddress);
                 out.println("</P>");
                 out.println("</BODY>");
                 ut.println("</HTML>");
              }  catch (SAXException se) {
                 getServletContext().log(se, se.getMessage());
                 System.exit(1);
              } catch (IOException ioe) {
                 getServletContext().log(ioe, ioe.getMessage());
                 System.exit(1);
              }    }
           } catch (IOException ioe) {
              getServletContext().log(ioe, ioe.getMessage());
           } catch (ModuleException me) {
              getServletContext().log(me, "Error handling request: "
              +  me.getMessage());
           } catch (Exception e) {
              getServletContext().log(e, e.toString());} }
```

Listing 27 (Continued) MyobServelet.java

```
// Error handler to report errors and warnings
   private static class MyErrorHandler implements ErrorHandler {
      /** Error handler output goes here */
      private PrintStream out;

      MyErrorHandler(PrintStream out) {
      this.out = out;
      }

/**
* Returns a string describing parse exception details
*/
      private String getParseExceptionInfo(SAXParseException spe) {
         String systemId = spe.getSystemId();
         if (systemId == null) {
            systemId = "null";
         }
         String info = "URI=" + systemId +
            " Line=" + spe.getLineNumber() +
         ": " + spe.getMessage();
         return info;
      }

// The following methods are standard SAX ErrorHandler methods.
// See SAX documentation for more info.

      public void warning(SAXParseException spe)
            throws SAXException {
         out.println("Warning: " + getParseExceptionInfo(spe));
      }

      public void error(SAXParseException spe)
            throws SAXException {
         String message = "Error: " + getParseExceptionInfo(spe);
         throw new SAXException(message);
      }

      public void fatalError(SAXParseException spe)
            throws SAXException {
         String message = "Fatal Error: "
               + getParseExceptionInfo(spe);
         throw new SAXException(message);
      }
   }
}
```

And here is the code for MyObReader.

Listing 27 (Continued) MyobServelet.java.

```
/*
* A simple SAX content handler that locates
* an email address in the COREid System
* SOAP response for a User Manager 'view' operation.
* This class extends DefaultHandler provided by JAXP package
* by adding start/end element and character handler methods to
* help in locating the data we are after, and an accessor method
* for our client to extract the data.
*
* Note:
*  This helper class makes assumptions, including
*  its intimate knowledge of the structure of its input.
*  If more than one email address is found, that information
*  is lost. This class is to illustrate the technique.
*  Requirements:
*     JAXP
*/
// APache XML packages
import org.xml.sax.*;
import org.xml.sax.helpers.*;
import java.util.*;
import java.io.*;
public class MyObReader extends DefaultHandler {
   private boolean inEmail = false;
   private boolean inEmailValue = false;
   private String theEmail = "no.email.address.found";
   // HANDLERS ////////////////////////////////////////////
   // parser calls this for each element in a document
   public void startElement(String namespaceURI, String localName,
         String rawName, Attributes atts)
         throws SAXException{
      if (localName.equals("ObEmail")) {
         inEmail = true;
      }
      if (inEmail && localName.equals("ObValue")) {
         inEmailValue = true;}
   }
   // parser calls this for each element in a document
   public void endElement(String namespaceURI, String localName,
         String rawName)
         throws SAXException{
      // Are we closing an ObEmail?
      if (inEmail && localName.equals("ObEmail")) {
         inEmail = false;
      }
      // Are we closing an ObEmail/ObValue?
      if (inEmailValue && localName.equals("ObValue")) {
         inEmailValue = false;} }
```

Listing 27 (Continued) MyobServelet.java

```
   // parser calls this for character content found inside
   // elements.
   // captures characters while inside an ObEmail/ObValue
   public void characters(char[] ch, int start, int length)
         throws SAXException{
      if (!inEmailValue) return;
      theEmail = new String(String.copyValueOf(ch,start,length));
   }

   // ACCESSORS ////////////////////////////////////////////
   public String getEmail() {
      return new String(theEmail);
   }
}
```

The servlet begins by establishing a connection to NetPoint (using a WebPass on a specified Webserver) using an HTTPConnection object. It then reads in the SOAP message that contains the request shown above from a file, and uses an HTTPClient object to POST the SOAP message to the server.

WebPass recognizes the request as SOAP message, and passes it on to NetPoint's IdentityXML processing logic interpretation. If the message is validated, the enclosed IdentityXML request is processed. The data resulting from the request and the response code are packaged and transmitted back to the servlet as a SOAP response. After verifying that the request generated a successful response code, the servlet creates and configures a JAXP SAX parser, passing in a custom content handler, MyObReader, which scans the data for the email address attribute.

If the document is successfully parsed, the MyObReader object provides the email address attribute value through its getEmail() method. In this example, it is assumed that the administrator has entered some text like green in the email address field.

The example servlet looks up this information, and uses it in generating the result HTML page. The value is inserted as the BGCOLOR attribute of the BODY tag in the HTML document. The page is returned showing the value in text, and the page background is green.

ObSSOCookie Example

The following example shows how to use Java to post a SOAP request (or make IDXML calls) to the COREid Server when it is protected using Netpoint. This example makes use of the obSSOCookie.

Note that you must have the Access Server SDK installed to create this type of request. See for “Installing the Access Server SDK” on page 1140 details.

Listing 28 SOAP Request Using the obSSOCookie

```
/**
 * This is a very simple SOAP example of how to invoke Oblix NetPoint
 * through SOAP.
 *
 * This program will make a soap request (send the request in soap.xml)
 * to the argument hostname:port/oblix/apps/corpdir/bin/corpdir.cgi
 *
 * Requirements:
 * *** ObSoapClient, a complete http client library from innovation, is
 *     required to run this test.  The software is free, and licensed under the
 *     GNU Lesser General Public License.
 *     HTTPClient is available at http://www.innovation.ch/java/HTTPClient
 *     This program has been tested with HTTPClient Version 0.3-2
 *
 * To run:
 * *** java ObSoapClient [-h hostname] [-p port] [-f inputfile] [-u oblixurl]
 *
 */

import java.net.URL;
import java.io.IOException;
import java.io.FileReader;
import java.io.BufferedReader;
import HTTPClient.CookieModule;
import HTTPClient.HTTPConnection;
import HTTPClient.HTTPResponse;
import HTTPClient.ModuleException;
import HTTPClient.NVPair;

public class ObSoapClient
{
   static String hostname = "sunlight.oblix.com";
   static String filename = "soap.xml";
   static int port = 80;
   static String oburl = "/identity/oblix/apps/userservcenter/bin/
userservcenter.cgi";

   public static void collectArgs(String args[])
   {
      for (int i = 0; i < args.length; i++) {
         if (args[i].equals("-h") && args.length >= i+1)
            hostname = args[i+1];
```

```
        else if (args[i].equals("-f") && args.length >= i+1)
           filename = args[i+1];
        else if (args[i].equals("-p") && args.length >= i+1)
           port = Integer.parseInt(args[i+1]);
        else if (args[i].equals("-u") && args.length >= i+1)
           oburl = args[i+1];
        else if (args[i].equals("-h") || args[i].equals("-help")) {
           System.out.println("Usage: java ObSoapClient [-h hostname] [-p port]
[-f filename] [-u oblixurl] \n");
        }
      }
   }

   /**
   * Read from soap.xml in current directory and return as string.
    */
   public static String getRequestFromFile()
   {
      StringBuffer data = new StringBuffer();
      try {
         BufferedReader reader = new BufferedReader(new FileReader(filename));

         for (String line = reader.readLine(); line != null;
            line = reader.readLine()) {
            data.append(line);
            data.append("\r\n");
         }
      } catch (Exception e) {
         System.out.println(e.toString());
      }
      return data.toString();
   }

   public static void main(String args[]) throws Exception
   {
      try {
         CookieModule.setCookiePolicyHandler(null);

         // initiate connection
         collectArgs(args);
         HTTPConnection con = new HTTPConnection(hostname, port);

         // collect response
         NVPair header[] = new NVPair[1];
         header[0] = new NVPair("Content-Type", "text/xml");
         HTTPResponse rsp =
            con.Post(oburl,
            getRequestFromFile(),
            header);

         // get status and act accordingly
         if (rsp.getStatusCode() >= 300) {
            System.err.println("Received Error: "+rsp.getReasonLine());
```

```
            System.err.println(new String(rsp.getData()));
         } else
            System.out.println(new String(rsp.getData()));
      } catch (IOException ioe) {
         System.err.println(ioe.toString());
      } catch (ModuleException me) {
         System.err.println("Error handling request: " + me.getMessage());
      } catch (Exception e) {
         System.out.println(e.toString());
      }
   }

}

/*
   NVPair form_data[] = new NVPair[2];

   form_data[0] = new NVPair("login", "J.Smith");
   form_data[1] = new NVPair("password", "J.Smith");
// form_data[2] = new NVPair("uid",
//    "cn=John Smith,ou=Corporate,o=Company,c=US");
// form_data[3] = new NVPair("program", "personPage");
// form_data[4] = new NVPair("tab_id", "Employees");

// HTTPResponse rsp = con.Post("/oblix/apps/corpdir/bin/corpdir.cgi",
form_data);

*/
```

4 Identity Event Plug-in API

The Identity Event Plug-in API enables you to extend the base COREid System functionality. This API provides a channel for COREid System data to flow between NetPoint applications and external software components. Applications for this API can be as simple as basic logging of NetPoint usage, or as sophisticated as data-filtering pipelines or seamless bridges to Enterprise Resource Planning systems.

The Identity Event Plug-in API is a standard installed component of the NetPoint product.

This chapter contains the following sections:

- "About the Identity Event Plug-in API" on page 172.
- "Connecting Events to Actions" on page 172 introduces the concept of *events* in the NetPoint data flow that can automatically invoke user-defined *actions* to change the outcome of user requests. Actions are associated with events by the content of a *catalog* file.
- "How the API Works" on page 182 describes the interface between COREid application events and actions.
- "Event Handling in the API" on page 196 describes the API event types and the functions you can build into each event type.
- "The API" on page 217 discusses the way in which actions load and execute, locates the library and header files that you use to create your own actions, and provides example files.
- "Cross-Application Support" on page 234 discusses a way the API can be used to pass information from one COREid application to another, for example to create a new user in the User Manager and add that new user to a Group.
- "Examples" on page 235 provides examples of source code to implement and configure two different types of action.

About the Identity Event Plug-in API

Just as a Web server can be configured to execute CGI programs and server-side scripts during the life cycle of an HTTP request, the Identity Event Plug-in API gives developers the ability to extend the COREid System by providing their own small applications, called *actions* or *event handlers*, to perform custom business logic and integrate with external systems. You connect event handlers to the events using a special configuration file named oblixpppcatalog.lst. NetPoint makes certain data available to the actions, which are then allowed to modify the data and influence the outcome of the event.

Examples of Uses of the Identity Event Plug-in API

Common uses of this API are for password validation, integration, and provisioning.

For example, in a password validation application, suppose a security architect recommends the use of 8-character passwords with 2-4 digits and 1-3 special characters. You can develop an event handler for Password Management events that use the Identity Event API and add this event handler to a COREid password policy.

As another example, suppose that new hires need to be recorded in a RDBMS to ensure that they receive a “Welcome to the company” packet. You could develop an event handler for the Enable step of each registration workflow instance to update the remote database using the RDBMS vendor’s API.

Finally, suppose that new users require a randomly generated unique ID to act as their login ID. You could develop an event handler for the Enable step of each registration workflow instance to generate a unique string in the required format and pass it back to the COREid System to use as the uid attribute value.

Connecting Events to Actions

This section describes actions and events in more detail, and explains how to connect them to each other using the Configuration File.

Types of Events

An event is a state change within the COREid system. Examples of events:

- A request was received and is about to be passed to the User Manager view program.
- Results have been generated by the Group Manager search program.

- A user has entered a challenge response while attempting a password reset.
- An attribute on a profile page for an Organization Manager tab has been modified.
- A workflow ticket awaiting approval by the corporate IT group has been approved.
- A user has entered a new password, and the password policy in force requires external validation.

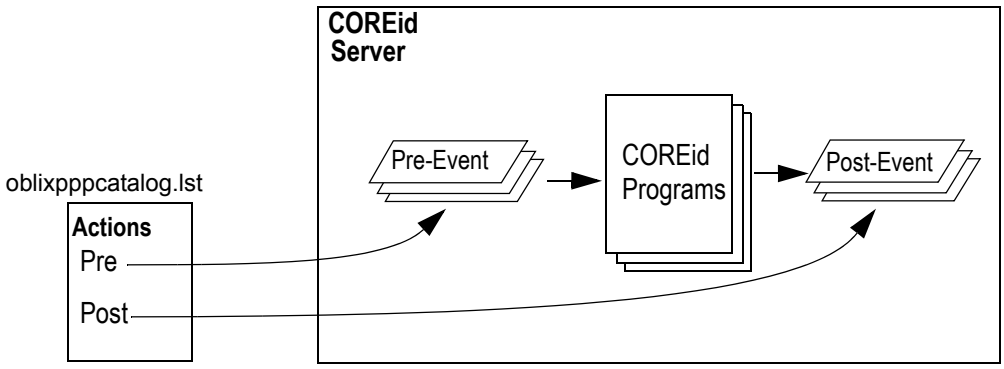
COREid provides functionality specific to five different types of events, summarized here. More detail for each type of event appears in the section on "How the API Works" on page 182.

COREid Program Events: Pre and Post

These are the most frequently used type of event. Each COREid application (User Manager, Group Manager, Organization Manager) contains a number of programs (view, search, and so on) that generate the displayed HTML for each page within the application. When any program runs, a pair of events is generated. Each of the programs recognizes this pair of events.

One event (Pre) is generated before the program begins to create the page. The Pre event allows an event handler to work with a request before it reaches a program. The other event (Post) is generated after the program has created the page, but before responding to the user with an HTML page. The Post event allows an event handler to work with the results of processing a request.

These two events are referred to as the *pre-processing event* and the *post-processing event* for that program, as shown in the following diagram.



OnChange

OnChange event interaction is provided as part of the set of NetPoint COREid applications (User Manager, Group Manager, and Organization Manager). Specifically, the OnChange event applies to the Profile page within each of these applications. When a change is made to any of the data in these pages, an OnChange event is generated. These events are triggered only after the changes are sucessfully committed to the directory.

Workflow Events

Workflows are definitions for a repeatable set of steps used to create or modify data. Workflow definitions are created and stored within COREid. The user can then reference the workflow by name, and instruct the *workflow engine* to process it when needed. Workflow steps each generate a pair of events (pre and post). The pre event allows an event handler to inspect and modify workflow data before the step is executed. The post event allows an event handler to inspect and modify workflow data after the step is executed. Workflow steps also generate an external action event.

The Pre, Post, and external action events in a workflow can process both LDAP data and template object data. This is a departure from the COREid applications, which only process LDAP data. The COREid System stores template attributes in fully qualified form on a workflow step, as follows:

attribute.class.domain

See the chapter on configuring template objects in the *NetPoint 7.0 Administration Guide, Vol. 1* for details.

Password Management Events

Password Management events are generated when an attempt is made to set the password of a user in a branch of the directory tree that is covered by a password policy whose external validation flag is enabled. Actions associated with Password Management events are used to check password quality against custom business rules.

As part of creating a password policy, you may enable the option “Externally specified validation rules.” NetPoint applies the password policy for the requester. If the requester is covered by the policy, then NetPoint checks to see if this flag is set. If it is, then NetPoint executes the password validation event which in turn carries out the action defined by the user. NetPoint also supports the Identity Event plug-in for the COREid Lost Password Management application. When you configure the oblixpppcatalog.lst file, the application name is lost_pwd_mgmt.

Lost Password Management

The event related to lost password management functionality is setChangedPassword, and the application name for this is lost_pwd_mgmt. The sample application name, event name, and action is lost_pwd_mgmt_setChangedPassword_pre.

Encryption Events

The COREid System applies a proprietary encryption method to several pieces of information. One is cookie information, such as the login cookie for a authenticated session. An encrypted version of this cookie is kept by the user's browser while a session is in effect. A second is the response half of the challenge/response pair used for Lost Password Management. The response phrase, to be given by the user in response to the challenge phrase, is stored encrypted in the directory. Password information is encrypted when included in a workflow. Encryption events are used to invoke user-defined encryption algorithms (implemented by actions) when COREid needs to encrypt a piece of data.

You can replace NetPoint's encryption technique with one of your own by adding actions to the Catalog to replace either or both of these default encryption methods. For example, you can replace NetPoint's default encryption scheme for cookies, challenge responses, and password fields in workflows using this method.

Types of Actions

An action is an event handler. More specifically it is a unit of external logic written by a NetPoint developer and then configured by a NetPoint administrator to execute in response to a particular event.

Actions have three formats: LIB, MANAGEDLIB, and EXEC.

Actions may perform their tasks without accessing external components, or they may use any available mechanism to access third-party applications and resources such as web services, RDBMS services, and ERP applications.

At startup time, the COREid Server reads its configuration catalog, which tells it what events have actions. When an event occurs, the server executes the associated action.

LIB Actions

A LIB action is a function within a shared library that the COREid Server calls. LIB actions reside in shared libraries on Unix or DLLs on Windows. Once dynamically loaded, the action function executes in the same process space as the COREid Server and has direct access through API functions to data objects held by the server.

For a LIB action, the COREid Server dynamically opens the shared library or DLL, locates the function that implments the action, and calls the function.

LIB actions have advantages. These are:

Fast loading—LIB actions are compiled binary objects that reside in shared libraries. They have relatively low startup overhead.

Reusable at runtime—LIB actions need only be loaded once. They then remain in virtual memory, ready for subsequent calls.

High performance—LIB actions execute quickly because they are binary code modules compiled from C or C++ source code. Of course, whether they are preceived as fast depends on the function they perform.

COREid data on demand—LIB actions have access to a great deal of data about the current request, the authenticated user, and other services from COREid via simple GET/SET API calls.

Scalable—LIB actions provide good scalability, even in high traffic applications, because they are simply functions that can be called repeatedly as requests are processed, with low overhead.

Disadvantages of LIB actions:

Limited support from third-party components—LIB actions, because they are written in C or C++ have relatively few freely available third-party APIs to call upon for external services such as RDBMS access, XML parsing and formatting, network services, cryptographyservices, LDAP services, and so on. These services are more widely available to the Java and PERL developer community.

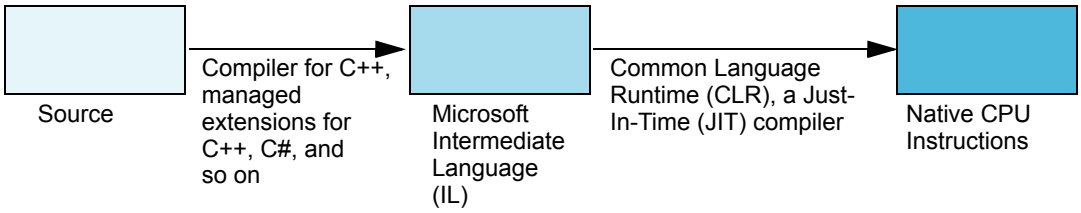
Specialist expertise required—LIB actions require more specialized skills to implement. This can increase the cost. For instance, even to deploy the same action on a Windows and Solaris environment simultaneously would require C/C++ development expertise in both platforms and development environments.

Platform-dependent source code—The steps necessary to author and build a shared library on Solaris are different from building a DLL for Windows on NT. Either defensive coding practices are required to ensure cross-platform source code, or multiple source trees must be maintained for a multi-platform deployment.

Potential to crash COREid Server—Any uncaught exceptions caused by errors in LIB actions will crash the COREid Server. This is because the action is running in the COREid address space, and if it accidentally causes a memory leak or memory trash, the server cannot detect and recover from this. These problems to not exist in EXEC actions because each EXEC action runs in its own address space and can only damage itself. The server can detect this because the child process exits without returning a success status.

MANAGEDLIB Actions

A MANAGEDLIB actions only run on Windows. A MANAGEDLIB action can be written in any .NET language. A .NET language is any source language for which a Microsoft Intermediate Language (MIL) compiler exists. MIL instructions are executed by the Microsoft .NET Common Language Runtime (CLR), which uses a just-in-time (JIT) compiler. The JIT compiler compiles the MIL instructions into native machine instructions. MIL instructions are compiled once and stored in dynamic memory. There is a modest performance hit the first time that managed code is executed.



MANAGEDLIB actions are similar to LIB actions. As with LIB actions, a MANAGEDLIB action is loaded into COREid memory. MANAGEDLIB actions also share most of the benefits of LIB actions.

In addition, MANAGEDLIB actions offer the benefits of managed code, including:

- **Language Choice**—You can write your plug-ins in VisualBasic, C#, Managed C++ (MC++), Java, or PERL.
- **Language Integration—**You can combine MIL modules compiled from different source languages into one assembly or plug-in. This provides the plug-in writer with a wider range of language choices for plug-in development.
- **Support for Memory Management**—The CLR provides garbage collection, freeing the plug-in writer from most memory management. The garbage collector will return memory to the heap when that memory is no longer referenced. However, the plug-in writer should ensure that there are no dangling references to objects. If there are dangling references, garbage collection will not occur for the unused memory.

- **.NET framework support**—The .NET framework SDK contains a wide range of functionality. This may reduce the need for third-party support in plug-in code.

EXEC Actions

An EXEC action is a standalone executable program that the COREid Server executes. EXEC actions reside in separate executables and run in their own process space. To process an EXEC action, the COREid Server starts a new child process and loads the executable passing its parameters. Input is streamed to the action on STDIN and output is received on STDOUT and the process's exit status.

- Communication with the COREid Server is limited to startup parameters and an XML stream for input, and an XML stream plus an exit status code for output. Any further access to the COREid data must be done like any other COREid client, using IdentityXML.
- Actions can also use any other APIs, such as an LDAP Identity Event Plug-in API to access directory information directly.
- For scripted EXEC actions, the action would be the interpreter, such as /usr/local/bin/perl, and the script itself would be passed as a command-line parameter.

Advantages of EXEC actions:

Choice of Development Languages—EXEC action developers can write the code in C, C++, Java, PERL, or any language that supports C-style command line processing and stdio.h compatible standard I/O processing.

Rapid prototyping—EXEC actions can be rapidly prototyped or developed using scripting languages such as PERL.

Platform-Independent Code—EXEC action source code can be platform independent because of the language neutrality. The same code written in PERL and Java will execute on Windows and Unix.

Java-Compatible—EXEC actions can be implemented in Java, giving them access to third-party services that only provide Java APIs.

Extensive Third-Party Support—EXEC actions cannot bring down the COREid Server. If they fail, the end user will see an error report and the COREid Server will continue to serve other requests.

Disadvantages include:

Poor Scalability—EXEC actions do not scale as well for high-traffic applications because a new child process is required per request.

Limited Access to COREid Data—EXEC actions get their input from command line parameters and from the (static) XML stream available on STDIN when they execute. There is no API to provide direct access to further COREid information. To do this, the action would have to implement an IdentityXML client and communicate with COREid Server over a separate connection.

XML Parser Required—EXEC actions need to parse XML for all but the simplest tasks in order to access their input. This means that they must have an embedded parser that understands the XML schema of any input they may receive. This adds to the startup time, the memory footprint, and the complexity of the action, and may be too heavyweight for many tasks.

Configuration File (Catalog)

NetPoint uses a configuration file, oblixpppcatalog.lst, to provide the link between NetPoint events to be responded to and custom actions to be taken. This file is called the Catalog. For LIB, MANAGEDLIB, and EXEC actions, this file is installed and *must stay* in the following directory:

COREid_install_dir/identity/oblix/apps/common/bin

Note: When you installed the NetPoint COREid System, you put it into an installation directory, for example: /usr/NetPoint/identity. As a convenient shorthand, this directory is called *COREid_install_Dir*.

Each entry in oblixpppcatalog.lst is a single line linking a COREid event to an action. Each line in the Catalog must contain at least five fields (six if you need to use the apiVersion field), delimited by semicolons. Each line must end with a semicolon. Lists of data items within each field are delimited by commas. Fields may be empty, indicated by the semicolons being next to each other. The precise content of each field varies with the action type and the kind of event to which it is responding.

The general form for a LIB or MANAGEDLIB entry is:

```
actionName;actiontype;;path;funcname;apiVersion;
```

With LIB and MANAGEDLIB actions, the *path* can be relative or a full path.

The general form for an EXEC entry is:

```
actionName;actiontype;netpointparam1,...;path;execparam1,...;apiVersion;
```

Fields within the entries are delimited by semicolons; Each entry *must* have at least five fields, and end with a semicolon, followed by a newline (carriage return and line feed).

Note: The special character # is used in this file to indicate lines that are comments. Do not use this character as part of a LIB or EXEC entry. It would be a mistake, for example, to call a LIB funcname getbuilding# since, for that entry, everything past the # would be ignored.

Explanation of each field is given in the following table. Read down each column to understand the content for each of the action types.

Field Name	LIB and MANAGEDLIB Actions	EXEC Actions
`actionName` (required)	Field 1. The action name. The name contains information that tells COREid which event type the action responds to and in some cases whether it should be performed before, as part of, or after the event.	Field 1. Same description as for LIB and MANAGEDLIB Actions.
`actiontype` (required)	Field 2. `managedlib` (This exact text) or `lib` (this exact text), depending on what type of action you are using.	Field 2. `exec` (This exact text).
`netpointparam1, . . .` (optional)	Field 3. This field is always empty.	Field 3. Used by EXEC actions only. The names of global parameters, delimited by commas. A table of these parameters is provided on page 193.
`path` (required)	Field 4. The location and name of the LIB or MANAGEDLIB file that implements the action.	Field 4. The location and name of the EXEC file that implements the action.
`funcname` (required)	Field 5. The name of *one* function to call from within the shared library, for the LIB or MANAGEDLIB action.	N/A

Field Name	LIB and MANAGEDLIB Actions	EXEC Actions

`execparam`
 	
`(optional)`
	N/A	Field 5. One or more input parameters to the EXEC action, delimited by commas.

`apiversion` (optional)	Field 6. Leave this field empty if the action is built using the NetPoint Version 6 API library. Otherwise, enter the exact text, preNP60, to tell the event handler that this action is based on a version of the API prior to NetPoint 6.0. This does not apply to MANAGEDLIB actions, since this feature is new in NetPoint 6.5.	Field 6. Same description as for MANAGEDLIB Actions.

Guidelines for Writing an Action

The procedure for creating an action is as follows.

Task overview: Writing an action

1. **Identify Requirements**—Investigate whether you need to validate or modify the inputs, results, or side effects of a COREid request or workflow in order to achieve results that the COREid System cannot deliver.
2. **Select the Event**—This depends on the following:
   a) **Availability**—The availability of the data.
   b) **Timing**—whether the system is in the desired state for the action when the event occurs
   c) **Performance**—To maximize performance, identify the least frequently used event that will yield the desired result.
   d) **Execution**—Determine if this action should run before (pre-event) or after (post-event) the request is processed by the COREid application.
3. **Write**—Write the action.
4. **Configure the Action**—A NetPoint administrator must edit the Identity Event plug-in API configuration catalog in

   COREid_install_dir//identity/oblix/apps/common/bin/oblixpppcatalog.lst

   The administrator enters an entry in the catalog to register the action and its parameters against a particular request. The administrator then restarts the COREid Server(s) or uses a portal insert to refresh the catalog of a running COREid Server.

How the API Works

The next section describes how actions are found and executed, from the NetPoint application's point of view. The following section, "NetPoint Applications, as Seen by Actions" on page 186, describes what happens and what data can be accessed, from an action's perspective.

Actions, as Seen by NetPoint Applications

The Catalog is loaded once, when the COREid System starts up. File content can be changed while the COREid System is running, but the changes take effect only if the file is reloaded. You can force changes to take effect by restarting the COREid System or by linking from any browser to the following URL:

```
http://hostname:port/identity/oblix/apps/admin/bin/
genconfig.cgi?program=flushCache&cacheType=ppp
```

For LIB and EXEC Actions—If you flush the Identity Event Plug-in (PPP) information from NetPoint, it "forgets" all it knew about DSOs and executables that contain actions. NetPoint reads the Catalog again when it next generates an event, and starts loading DSOs on demand, depending on what actions are configured and what events occur.

For MANAGEDLIB Actions—The DSO (in managed code terminology, this is the assembly or DLL) is loaded once into the default application domain. If a plug-in writer rebuilds the assembly, they will need to restart the COREid Server to ensure that the new assembly is loaded the first time an action from that assembly is invoked.

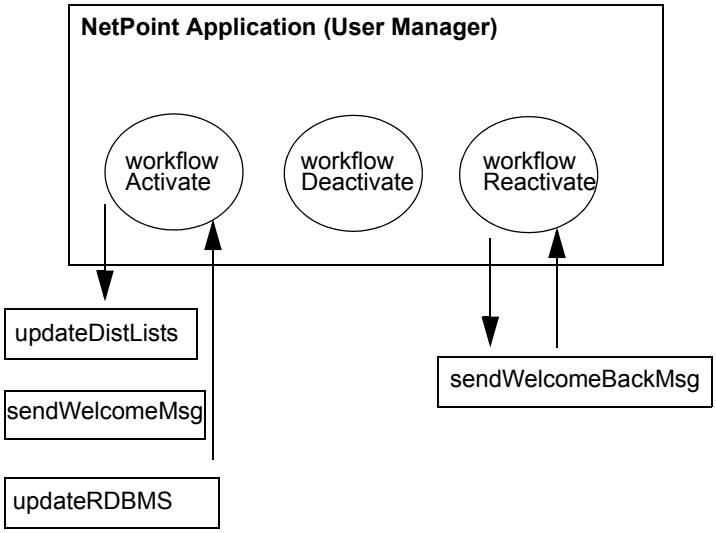
Multiple actions can be defined for a single event. If multiple actions are defined, all of the actions are performed, in the order that they appear in the Catalog. This approach allows you to build action *pipelines*, where the output of one action can become the input of the next.

Keep in mind, however, that for a typical event, the event that invoked the action was caused by user activity. While your code is processing the data, and passing its output down the chain, the end user may be waiting for a result. You should include the impact of *user-perceived* responsiveness in the design and testing of all actions, especially if multiple actions are expected for a single event. Note also that if any of the multiple actions returns an error, additional actions in the pipeline for that event instance will *not* be performed.

The following diagram shows how three possible events can be configured with actions for the User Manager application.

Assume that the workflowActivate event is associated in the Catalog with three custom actions to be performed during post-processing:

- Invoke an action to extract information about the newly activated user, and add the user to default company email distribution lists.
- Send a pre-written email message template to relevant lists and individuals (possibly based on NetPoint data) welcoming the new user to the organization.
- Trigger an external business process that updates each of various external company databases with appropriate information about the user. This can be as simple as an application that pushes the new user's information to a table where external programs can pick it up.



In this example, when a user request generates the workflowActivate event, User Manager consults the Catalog, determines that this event has no configured pre-processing actions, and proceeds to generate the page in XML. It then checks for post-processing actions and finds three: updateDistLists, sendWelcomeMsg and updateRDBMS. User Manager checks the Catalog to see whether the first action, updateDistLists, is a LIB or EXEC action. How processing proceeds depends on the result of this test:

- **For a LIB Action**—User Manager dynamically loads the DSO containing the function (if it is not already loaded) and obtains a pointer to the function within the action. It then calls the function, passing the name of the event for which it is being invoked and a pointer to an ObPPPData object through which the action can interact with User Manager. The action performs its tasks, querying User Manager as needed through calls to ObPPPData methods. When its task is complete, the action function returns status information that the event uses to decide its next behavior.
- **For a MANAGEDLIB Action**—User Manager dynamically loads the DSO containing the function if it is not already loaded and obtains a pointer to an object that implements the IPPPData interface. NetPoint will reference an

EventAPI object and invoke the “action” method on that object, passing a reference to IPPPData as a parameter of the “action” method. The EventAPI object is a singleton, meaning that the first request will instantiate the object and subsequent requests will use this object. The action performs its tasks, querying User Manager as needed through calls to IPPPData methods. When its task is complete, the action function returns status information that the event uses to decide its next behavior.

Note: In managed code terminology, DSO is referred to as an assembly or a dll.

- **For an EXEC Action**—User Manager starts up the executable in a new process, making connections to its STDIN and STDOUT streams. The argv[] array of command line arguments is built as well. The first argument is the total count of arguments. The last is always the data for the set of NetPoint parameters specified in the catalog (if any), provided in a specific XML format, called *EventXML*. The arguments in between match the values given for EXEC parameters in the catalog file (if any). User Manager always sends the XML data representing the current state of the request, in EventXML format, to the action on its STDIN. The action interprets its arguments, if any, reads its STDIN and performs its task, which may or may not involve XML parsing to extract and/or replace information received from User Manager. When complete, the action optionally writes the XML data out on its STDOUT. The action is not required to return the XML data, because NetPoint keeps the original version as the default, in parsed form. (If the data is large, avoiding an extra parsing operation can be worthwhile).

Note: If any action modifies the XML data, it is the action’s responsibility to make sure the output XML conforms to the appropriate XML Schema. How to find the correct schema file for the event you are handling is described in “Connecting Events to Actions” on page 172.

When User Manager receives the result status code, either from the return value of a LIB action function or from the exit status of a terminating child process that was running an EXEC action, it proceeds as follows:

- **STATUS_PPP_OK**—User Manager looks for the next post-processing action in the Catalog that is configured for workflowActivate. If there is one, User Manager goes through the preceding procedure again for that action, passing in the possibly modified XML data received from the earlier actions.
- **STATUS_PPP_ABORT**—The action has signalled an error. In this case, User Manager does not even check for subsequent actions for the event. It translates any information transmitted by the action through the API into an error, and handles the error. This result generally results in an error message being displayed to the user, who can then report the problem.

- **STATUS_PPP_WF_ASYNC**—For use with workflows, this result tells the event to wait for an asynchronous, probably manual, action to be completed. For example, this might ensure that a currently unavailable database is updated before the workflow continues. See “Workflow Events” on page 205 for more details.
- **STATUS_PPP_WF_RETRY**—For use with workflows, this response tells the event that the workflow step did not complete, probably because of invalid data, that therefore needs to be reentered. The event increments its retrycount, and starts again.

In the example, workflowActivate has three post-processing actions. You can view this as a pipeline, because the data flows from one action to the next. In practice, each action returns to User Manager before the next is called. If all of the actions return STATUS_PPP_OK, User Manager completes its processing for the request, applying the XSL stylesheet for the page and returning the resulting HTML page to the browser.

No actions are configured for the workflowDeactivate pre- or post-processing events, so User Manager builds the result page without calling any actions.

The workflowReactivate post-processing event is configured to call the sendWelcomeBackMsg action. User Manager calls this single action and applies XSL processing to its output before returning the result page to the user.

Note: The action has direct access only to data known to the current event. For example, in order to access full user information from the directory, the action might have to communicate with NetPoint using a different method, such as the IdentityXML interface, before it has sufficient information to accomplish a given task.

The preceding illustration uses User Manager as an example; the other COREid System applications that generate events behave in exactly the same way with respect to the Identity Event Plug-in API.

For detailed examples of LIB, MANAGEDLIB, and EXEC actions, including sample code and Catalog entries for configuration, see “Examples” on page 235.

NetPoint Applications, as Seen by Actions

Following topics provide details about NetPoint applications and actions:

LIB Actions

LIB actions are only available in C or C++. However, if you want to write these actions in Java, you can write a JNI to wrap C++ functions with the Java API.

LIB Interface

The LIB action interface to NetPoint is defined by the ObPPPData class provided in the obpppdata.h. file. See "Development Environment" on page 229 for the location of this file. ObPPPData defines five methods that a LIB action may use to access NetPoint data. These methods are:

- **Get**—Use this method to get the value or values (attributes may be multi-valued) for a specified parameter from the NetPoint application that triggered the event. The method returns a pointer to an array of values matching a key. The key is any attribute that is known to the application handling the event. The key may also be one of the Global parameters; see "Global Parameters" on page 193. The last member of the array is a NULL.

```
virtual const char * const *Get(
  const char *key) const;
```

Responsibility for allocating and freeing memory for the return value lies with NetPoint. If you request a value for a parameter that is not valid for the event triggering the action, only the NULL value is returned.

- **Set**—Use this method to set the value(s) for a given parameter to be sent to the NetPoint application. You should set values only for those parameters that are valid for the event.

```
virtual int Set(const char *key,
     const char * const *value) = 0;
```

Key represents the parameter to be set; value is an array of values to be used. The last member of the array must contain a NULL.

Responsibility for allocating and freeing memory for the input parameter content lies with the API developer.

COREid returns 1 if the set is successful, 0 if not.

- **Receive**—Use this method to request and receive event data from the NetPoint application as an XML string. You will need to understand the structure of this XML string in order to locate data within it. See “Working with XML” on page 194 and the chapter on PresentationXML in the *NetPoint 7.0 Customization Guide*.

```
virtual const char *Receive() const = 0;
```

- **Send—**Use this method to send replacement content for the event page to the NetPoint application as an XML string. Generally, this will be EventXML, from which the COREid application extracts the information it needs. In the case of post actions, however, this is expected to be PresentationXML. This PresentationXML string completely replaces the output that the NetPoint application would otherwise have used to generate an HTML page for the user or would have passed to the next action if this action is part of a pipeline. Your new content may differ in a minor way, such as the addition of a copyright or other text message, or it may contain significantly different data.

```
virtual void Send(const char *data) = 0;
```

The XML data string being sent must match the schema expected by the application receiving it. Best practice is to verify the XML data against the schema with an XML editor, before using the action in a live environment.

- **SetResultString**—Use this method to set the content of a result string to be displayed to the user by the NetPoint application. The exact manner in which the text is then shown varies with the event to which the action is responding.

```
virtual void SetResultString(const char *str) = 0;
```

str contains the text to be displayed.

Note: This method cannot be used for post events, because the data returned by them is in PresentationXML format. If an error is to be displayed, the developer is responsible for building it into the PresentationXML.

Load Behavior

The Dynamically Shared Object (DSO) for a LIB action is loaded into NetPoint's address space when it is first needed and remains there. A new version of the DSO can be generated and installed while NetPoint is running, but any revised actions contained in the DSO will not be loaded unless the COREid System is stopped or started, or the loaded Catalog is flushed using the URL described in "Configuration File (Catalog)" on page 179. If the file is flushed, then the action is reloaded the next time its corresponding event occurs.

Functions for LIB actions are loaded as needed into NetPoint applications, and executed directly by them. For this reason, you will need to link with the Oblix-provided interface library on Windows platforms or with the runtime shared library itself for UNIX. See "Development Environment" on page 229 for the location of the Windows library.

LIB Examples

LIB Code examples can be found installed under

COREid_install_dir/oblix/unsupported/ppp/ppp_dll

and are provided under "Examples" on page 235.

MANAGEDLIB Actions

MANAGEDLIB actions can be written in any language supported by the Microsoft .NET framework for managed code, including Visual Basic, C# and C++.

MANAGEDLIB Interface

The MANAGEDLIB action interface to NetPoint is defined by the IPPPData interface. See "Development Environment" on page 229 for the location of the header file. The header was written in MC++. The IPPPData interface will be syntactically different in other .NET languages, but will work the same way, that is, the semantics will be identical.

IPPPData defines five methods that a MANAGEDLIB action may use to access NetPoint data. The definitions for these methods are similar to those for LIB actions:

- **Get**—Gets value or values for a specified parameter (argument key) from the NetPoint application that triggered the event.

```
String * Get( String * key ) __gc[];
```

- **Set**—Set the value(s) for a given parameter (argument key) to be sent to the NetPoint application. You should set values only for those parameters that are valid for the event.

```
int Set( String * key , String * value __gc[] );
```

- **Receive**—Use this method to request and receive event data from the NetPoint application as an XML string. You will need to understand the structure of this XML string in order to locate data within it. See "Working with XML" on page 194 the chapter on PresentationXML in the *NetPoint 7.0 Customization Guide*.

```
String * Receive( );
```

- **Send—**Use this method to send replacement content for the event page to the NetPoint application as an XML string.

```
void Send( String * data );
```

  The XML data string being sent must match the schema expected by the application receiving it. Best practice is to verify the XML data against the schema with an XML editor, before using the action in a live environment.

- **SetResultString**—Use this method to set the content of a result string to be displayed to the user by the NetPoint application. The exact manner in which the text is then shown varies with the event to which the action is responding.

```
void SetResultString( String * str );
```

  str contains the text to be displayed.

Note: This method cannot be used for post events, because the data returned by them is in PresentationXML format. If an error is to be displayed, the developer is responsible for building it into the PresentationXML.

Load Behavior for MANAGEDLIB

The DSO (managed assembly or dll) is loaded once into the default application domain, which is part of the NetPoint process. If a plug-in writer rebuilds the assembly, they will need to restart the COREid Server to ensure that the new assembly is loaded the first time an action from that assembly is invoked.

MANAGEDLIB Examples

MANAGEDLIB code examples can be found installed under

COREid_install_dir\oblix\unsupported\ppp\dotnet\managedcplusplus

and are provided under “Examples” on page 235.

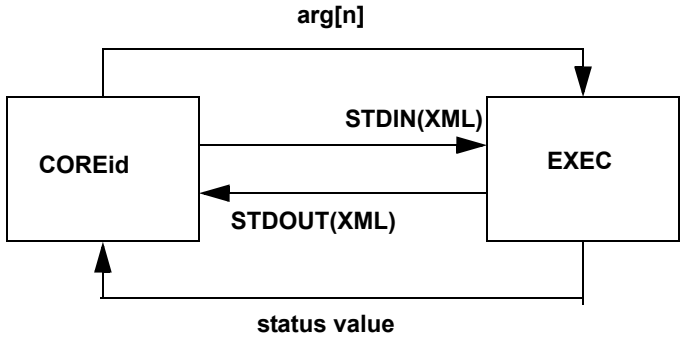
MANAGEDLIB Actions

Using Windows-based Managed Code, you can write MANAGEDLIB actions in Visual Basic, C#, C++, and any other language that uses Managed Code. Managed Code is only appropriate for MANAGEDLIB actions, not for EXECs. This is because MANAGEDLIB actions are loaded into COREid memory.

EXEC Actions

Exec Interface—Executables run as distinct processes and do not share an address space with NetPoint.

The NetPoint program determines the data to be sent and received between itself and the executable, based on the NetPoint and Executable parameters specified in the Catalog entry. The action then receives a set of command line arguments and XML data representing the event on STDIN. The EXEC action returns a status and, optionally, XML data on STDOUT. The possibilities are shown in the following diagram:



The set of command line arguments has a fixed logical structure. Consider, for example, the argv[] array of command line parameters. The first member of this array is the total count of arguments. The last array member is always the data for the set of NetPoint parameters (if any) specified in the catalog entry for the event, always provided in EventXML format. (These are the same set of parameters available to LIB actions; see the full list at "Global Parameters" on page 193.) The arguments in between are the values given for EXEC parameters (if any) in the catalog entry for the event. The exec parameters are user-defined instructions to the EXEC action that control its operation.

The format of the XML data sent to the action and returned by it varies with the type of event. In most cases, both the STDIN and STDOUT data will be in EventXML format. Post processing events are an exception; they *always* return PresentationXML, as discussed in greater detail in "Pre and Post Events" on page 198.

EXEC actions are able to get and return the same data as LIBs, but do it in a more complex way, generally requiring parsing of the XML data. Here are the equivalences:

- **Equivalence to Get**—The desired value must be obtained by first locating the attribute name in the EventXML or PresentationXML string, then extracting the value.

  Values for global parameters are provided, in EventXML format, as the last command line argument.

- **Equivalence to Set**—The user must start with the full EventXML or PresentationXML string, then locate the attribute name in the XML and insert the value.

- **Equivalence to Receive**—This is the EventXML or PresentationXML string for the event, which is always provided to the EXEC using its STDIN.

- **Equivalence to Send**—This is the EventXML or PresentationML which the action optionally returns on its STDOUT.

- **Equivalence to SetResultString**—Use ObResultString to name the ObParam and provide the message string as its value.

Load Behavior

Unlike LIB actions (which are cached) EXEC actions are executed afresh from the file system each time they are used. This means that they can be replaced at any time with a new version. The new version is executed the next time the corresponding event is triggered.

EXEC Examples

EXEC Code examples can be found installed under

*COREid_install*_dir/oblix/unsupported/ppp/ppp_exec

Global Parameters

A special set of NetPoint global parameters can be retrieved. Lib actions get values for these parameters interactively, using the **Get** method. Exec actions get values by providing one or more of the parameter names in the NetPoint parameter list. Either way the developer specifies a predefined fixed value parameter name, also called a *key*, as listed in the following table. Use the full uppercase parameter name shown in the table, preceded by either ObRequest or ObEnv, as shown.

User Identity Key Name	Description of Data
`ObRequest.REMOTE_ADDR`	Client IP address, for example, 666.777.888.999. This is the IP address of the user making the request.
`ObRequest.REMOTE_HOST`	Client's DNS address (for example, www.foo.com).
`ObRequest.REMOTE_PORT`	The port number at which the client host is listening.
`ObRequest.REMOTE_USER`	Name of the HTTP authenticated user.
`ObRequest.HTTP_USER_AGENT`	Name of the client's browser (for example, Mozilla/4.07).
`ObRequest.OBLIX_AUTH_USER`	The NetPoint authenticated user.
`ObRequest.TARGET_UID`	UID for the target entry.
`ObRequest.<any requestInfo parameter>`	A requestInfo parameter is any of the values that appear in the URL for a displayed page, following the delimiters $ or &.
`ObEnv.INSTALL_DIR`	Installation Directory for the COREid Server.

For example, to get the name of the user making the request from within a LIB or MANAGEDLIB Faction, the content could be:

```
currentuser = Get("ObRequest.OBLIX_AUTH_USER")
```

The catalog entry for the equivalent EXEC request might look like this:

```
userservcenter_view_pre;exec;
ObRequest.OBLIX_AUTH_USER;
../../../unsupported/ppp/ppp_exec/pppexectest;
someinstruction;
```

Working with XML

Topics are presented as follows:

Event XML Format

EventXML provides a standard, predictable format for use by LIB, MANAGEDLIB, and EXEC actions. The schema for EventXML looks like this:

```
     <?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="http://www.oblix.com/"
xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://www.oblix.com/"
elementFormDefault="qualified">
             <xs:element name="ObEventParams">
                <xs:complexType>
                <xs:choice minOccurs="0"
                     maxOccurs="unbounded">
                     <xs:element name="ObParamList"
                     minOccurs="0"maxOccurs="unbounded">
                        <xs:complexType>
                           <xs:sequence>
                              <xs:element ref="ObParam"
                              maxOccurs="unbounded"/>
                           </xs:sequence>
                           <xs:attribute name="name"
                           type="xs:string" use="required"/>
                        </xs:complexType>
                     </xs:element>
                     <xs:element ref="ObParam" minOccurs="0"
                        maxOccurs="unbounded"/>
                </xs:choice>
                </xs:complexType>
             </xs:element>

   <xs:element name="ObParam">
                <xs:complexType>
                   <xs:sequence>
                      <xs:element name="ObValue" type="xs:string"
                         minOccurs="0" maxOccurs="unbounded"/>
                   </xs:sequence>
                   <xs:attribute name="name" type="xs:string"
                   use="required"/>
                </xs:complexType>
   </xs:element>
</xs:schema>
```

Consider an example. Suppose the Catalog entry for an EXEC action is as follows:

```
        userservcenter_view_pre;exec;
ObRequest.cn,ObRequest.sn;
../../../unsupported/ppp/ppp_exec/pppexectest;
execparam;
```

This example specifies that the EXEC action pppexectest is to be invoked before (pre) NetPoint begins to build the person profile page (view) in User Manager (userservcenter). Information is requested for the cn parameter and sn parameters. The executable parameter execparam is to be included as the first command line argument to the executable.

The last argument of the command line information passed to the EXEC action, containing the EventXML, will be as shown in the following listing. Note there are as many instances of ObParam as there are requested parameters.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<ObEventParams
 xmlns="http://www.oblix.com/">
        <ObParamList name="ObRequest">
             <ObParam name="cn">
<ObValue>John Smith</ObValue>
             </ObParam>
<ObParam name="sn">
   <ObValue>Smith</ObValue>
        </ObParam>
</ObParamList>
</ObEventParams>
```

PresentationXML Format

The content of PresentationXML is highly variable, because COREid allows the user to modify the appearance and content of screens to satisfy site-dependent requirements. An explanation of the XML content and structure is provided in the "PresentationXML" chapter of the *NetPoint 7.0 Customization Guide.*

Parsing XML

In order to work with either EventXML or PresentationXML, the developer will need to be able to parse the XML data stream, to locate the points in the stream where data is provided (to in effect **Get** data) or must be inserted (to in effect **Set** data).

We do not attempt here to tell developers how to program such a parser. However, a set of examples is provided at:

COREid_install_dir/oblix/unsupported/ppp/parser_test

See "Parser Example Files" on page 233 for a list of these files.

The files assume the developer is using the free Apache XML parser, XERCES, source code for which can be obtained from:

`http://xml.apache.org/`

The content of the EventXML and PresentationXML strings technically *is* predictable, but depends in very complex ways upon which event is occurring and in which application. The recommended approach is to set up an action that returns the XML stream for the desired application and event combination, and capture the stream in a file. Then, code the action to work with that information. This approach is particularly appropriate for actions to be written for post-processing events, when the stream consists of highly variable PresentationXML information.

Event Handling in the API

For each of the five event types, this section describes:

- The syntax required in order to link that event type to an action in the Catalog.
- The functions required for initialization and shutdown of the event handler.
- The valid interface methods for the event type, and the content of the interface data.
- The set of status responses, one of which the action *must* return to the application.

For details see:

- "Event Handler Initilization and Shutdown Functions" on page 197
- "Pre and Post Events" on page 198
- "OnChange Events" on page 202
- "Workflow Events" on page 205
- "Password Management Events" on page 213
- "Encryption Events" on page 215

Note: Events are described in greater detail at "Types of Events" on page 172. All event types support LIB, MANAGEDLIB, and EXEC actions.

Event Handler Initilization and Shutdown Functions

For LIB and MANAGEDLIB actions, Pre and Post Processing (PPP) provides functions for initialization and shutdown for the event handler. These initialization and termination functions are called each time when the PPP library is loaded or unloaded.

For MANAGEDLIB actions, you must define a singleton object of class EventAPI. The constructor will be invoked upon loading the DSO, while the destructor will be called when the default application domain is unloaded from the NetPoint process during shutdown. The initialization code must be placed in the constructor for this class, and the shutdown code must be placed in the destructor. These replace the Init and Term functions provided for LIB and MANAGEDLIB actions.

Global initialization and cleanup should be done only within the functions shown below.

ObInitEventAPI ( )

For LIB, actions, the ObInitEventAPI () function is called when the DSO is loaded. This function, shown below, provides all global initialization such as reading of configuration files and creation of log files.

```
unsigned int OBLIX_DLLEXPORT ObInitEventAPI (void)
```

This function is guaranteed to be called in a thread-safe manner.

If this function is present in the plug-in, NetPoint calls it after the plug-in has been loaded.

Return Values

The function can return either of the following two response values, which are defined in the obppp.h. file:

- **STATUS_PPP_OK**—This is the result the function should return if it succeeds. This function result tells the application that the function has completed execution without error.
- **STATUS_PPP_ABORT**— This is the result the function should return if it fails. This function result tells the application that the function failed to complete execution because of an error. Subsequent calls to the plug-in will not be made.

ObTermEventAPI ( )

For LIB actions, the function ObTermEventAPI () is called when the DSO is unloaded. This function, shown below, performs clean-up activity such as releasing any allocated memory and closing any opened files.

```
unsigned int OBLIX_DLLEXPORT ObTermEventAPI (void)
```

This function is guaranteed to be called in a thread-safe manner.

If this function is present in the plug-in, NetPoint calls it when the server is being shut down.

Return Values

The action returns the following value which are defined in the pppdlltest.cpp file:

STATUS_PPP_OK—The function returns this result to tell the application that it has completed the action without error.

STATUS_PPP_ABORT—The function returns this result to tell the application that it has encountered an error.

Pre and Post Events

See the following for details:

- "Catalog Entry" on page 198
- "Interaction Methods" on page 201
- "Return Values" on page 202

Catalog Entry

For this event type the format for the entry in the Catalog for a LIB action or a MANAGEDLIB action is:

```
actionName;lib;;libname;libfuncname;apiVersion;
```

For EXEC actions the format for the entry is:

```
actionName;exec;NPparam1,...;execname;execparam1,...;apiVersion;
```

Note the punctuation within each entry. Fields are separated by semicolons; lists of items within fields are separated by commas. The entry is terminated with a semicolon. Fields may be empty. The following table describes each field in detail.

Field Name	Description
actionName (for events in Group Manager, Organization Manager, and User Manager)	Required. Provide this information in the form APPNAME_EVENTNAME_PPPTYPE. Note the underscores used to separate the three parts. APPNAME is the NetPoint application name. Valid application names are: **groupservcenter**—or Group Manager **objservcenter**—for Organization Manager **userservcenter**—for User Manager EVENTNAME is one of the possible events for the APPNAME application, as described in “Oblix NetPoint Events” on page 609. PPPTYPE is one of two values: **pre**—means the action is a preprocessing action, to take place before the event **post**—means a post-processing action, to occur after the event.
(other fields)	See the descriptions in “Configuration File (Catalog)” on page 179.

The following are some LIB action Catalog entry examples. For the examples shown here and in following sections, file entries are shown with line breaks at the semicolon delimiters, to allow printing in this Guide. In the actual file, the content must be entered all on one line. Also, source for many of the example actions is provided as part of the NetPoint installation. See “Development Environment” on page 229.

```
userservcenter_view_pre;lib;;
../../../unsupported/ppp/ppp_dll/libppp_dll.dll;
PreProcessingTest;;
```

For a MANAGEDLIB action Catalog, the entry would be as follows:

```
userservcenter_view_pre;managedlib;;
c:\unsupported\ppp\ppp_dll\libppp_dll.dll;
PreProcessingTest;;
```

This example calls for the PreProcessingTest action function in the ppp_dll.dll library to be performed before (pre) the person profile page (event = view) is built by the User Manager (application = userservcenter). Because the file type is lib, this is a LIB action implemented by a function, meaning that the DSO must be loaded and the function located within the DSO before the action can be performed.

This example action that is provided in the unsupported directory changes the requested uid value to the following DN:

```
cn=Pick Carli,ou=Customer10K1,ou=Customers,o=Company,c=US
```

Mr. Smith's profile is always displayed when the user requests a profile page in the User Manager, regardless of the uid for which the request was made.

```
userservcenter_view_post;lib;;
../../../unsupported/ppp/ppp_dll/ppp_dll.dll;
PostProcessingTest;;
```

This example configures the PostProcessingTest action function in ppp_dll.dll to be invoked after (post) the building of User Manager's person profile (view) page. This example action, which is provided in the unsupported directory, forces a message to be displayed instead of the profile page.

Further examples of Catalog entries can be found in the default Catalog file located at:

COREid_install_dir/oblix/apps/common/bin/oblixpppcatalog.lst

Interaction Methods

Get

Operation	User Identity Key Name	Description of Data
GET	<attribute name>	For a LIB action, returns a NULL-terminated array holding each of the values provided for the named attribute within the XML data used to create the display. For managed code, returns an object of base type System.Array whose contents can be enumerated.

Set

Operation	User Identity Key Name	Description of Data
SET	<attribute name>	Sets values for the named attribute within the XML data used to create the display.
SET	ObRequest.<any requestInfo parameter>	Sets the single value for any of the RequestInfo parameters that appear in the URL for a displayed page, following $ or &.

Receive

XML data received in response to this request can be of two different formats, depending upon whether the action is pre or post. Pre actions receive data in the EventXML format, as described at “Working with XML” on page 194. Post-processing actions receive data in the PresentationXML format, as described in the “Presentation XML” chapter in the *NetPoint 7.0 Customization Guide*.

Note: In the case of pre actions, the EventXML will contain values only if the event being monitored is one that allows a change of attributes.

Send

XML data sent back to the application must be the same type as was received. The data sent to the application must conform to the formal schema for each type of data or else it will be rejected. The section “Parser Example Files” on page 233 lists some files provided with the NetPoint installation, which can be used to verify the data using an XML editor.

SetResultString

A string returned with this method will be displayed by the COREid application.

Return Values

The action should return one two response values, which are defined in the Obppp.h file:

- **STATUS_PPP_OK**—This is the *success* response. The action sends this value to tell the application that the action has completed without an error. Value = 0x00h.
- **STATUS_PPP_ABORT**—This value returned means an error has occurred. Value = 0x01h.

Failure to formally return a response value will cause unpredictable behavior in the application, depending upon the default return value that the operating system will supply instead.

OnChange Events

See the following for more information:

- “Catalog Entry” on page 203
- “Interaction Methods” on page 204
- “Return Values” on page 205

Catalog Entry

Entries to the Catalog for this action type are the same as for pre- and post-processing events, except for the action name.

Field Name	Description
`actionName` (for OnChange)	Required. Provide this information in the form APPNAME_STRUCTURALCLASSNAME _onchange. APPNAME is the NetPoint application name. STRUCTURALCLASSNAME is the name of the structural class that contains the attribute whose change in value is to be monitored. If the attribute belongs to an auxiliary class then the name of the structural class it is attached to. onchange precisely identifies the type of action.
(other fields)	See the descriptions in “Configuration File (Catalog)” on page 179.

This event gets triggered only after the changes are successfully committed to the directory.

Here is an example:

```
     userservcenter_inetOrgPerson_onchange;lib;;
..\..\..\unsupported\ppp\ppp_dll\ppp_dll.dll;
uscOnChange;;
```

This example calls for the action `uscOnChange` to monitor any changes that the event makes to attributes belonging to the `inetOrgPerson` class, or to any of its attached auxiliary classes.

Interaction Methods

Get

Operation	User Identity Key Name	Description of Data
GET	<attribute name>.ObOldValue	The old value for the attribute.
GET	<attribute name>.ObNewValue	The new value for the attribute.
GET	<attribute name>.ObChangeType	The type of change that was made. Possible values: **OB_ADD**—A new value was added. **OB_MODIFY**—An existing value was changed. **OB_DELETE**—An existing value was deleted. **OB_NOCHANGE**—The value was not changed.

Set

This method is not supported for the OnChange event.

Receive

Receive() is used to get data from COREid. XML can be received just as for pre- and post-processing events, but only in EventXML format.

Send

The Send() method is used to send data to the COREid. The onChange event handler is called in response to completion of an operation that changed data in the Profile page of a COREid application. For the onChange event handler, you use the Send() method to set a message to be displayed on the screen after the modification operation completes execution. A call to the Send() method must be sent in EVENTXML format.

Send() method is for sending data back to COREid. For the onChange event, Send() can only be used to set a message to be shown after the operation is completed. The same result can be implemented using the SetResultString() call

SetResultString

A string returned with this method will be displayed by the COREid application.

Return Values

The action *must* return one of two response values, which are defined in the Obppp.h file:

- **STATUS_PPP_OK**—This is the *success* response. The action sends this value to tell the application that the action has completed without an error. Value = 0x00h.
- **STATUS_PPP_ABORT**—This value returned means an error has occurred. Value = 0x01h.

Failure to formally return a response value will cause unpredictable behavior in the application, depending upon the default return value that the operating system will supply instead.

Workflow Events

See:

- "Catalog Entry" on page 206
- "Interaction Methods" on page 208
- "Tables of Workflow Attributes" on page 209
- "Return Values" on page 212

Catalog Entry

Workflow entries to the Catalog use the same format as for pre- and post-processing events. Also, except for the actionName field, the table describing the fields within entries is identical. For a description of the format and requirements for the fields of the workflow entry that are common to all entries, see “Configuration File (Catalog)” on page 179.

Field Name	Description
`actionName (for workflows)`	For events triggered by workflows, use the form WORKFLOW-DEFINITION-ID_ STEP-DEFINITION-NUMBER_ PPPTYPE. Note the underscores used to separate the three parts. WORKFLOW-DEFINITION-ID is the unique identifier used to label the workflow. The DN for the workflow is shown in the workflow definition view. Ssee the *NetPoint 7.0 Administration Guide Volume 1*. STEP-DEFINITION-NUMBER is the number of the step within the workflow. PPPTYPE is one of three values: preaction—The action is a preprocessing action, to take place before the workflow step. externalaction—Means the action occurs as part of the workflow step. The workflow waits for this action to complete before continuing. postaction—Means a post-processing action, to occur after the workflow step.
(other fields)	See the descriptions in “Configuration File (Catalog)” on page 179.

Here is an example of a workflow event entry in the `oblixpppcatalog.lst` Catalog file:

```
63f004504f83455b924133acd0ef2e87_3_externalaction;
   lib;;../../../unsupported/ppp/ppp_dll/libppp_dll.so;
   WorkflowExtActionTest;;
```

This example calls WorkflowExtActionTest as an externalaction during step three of the workflow whose ID is `63f004504f83455b924133acd0ef2e87`.

Example: Calling Logger as a Post-Processing Action After a Workflow Step

Here is another example of an entry in the Catalog. This example calls the Logger action after Step 1 of the workflow completes execution. The workflow ID number—2e22c064723e4030a05b437e059fe4d6—is used to identify the workflow that contains the step. The full entry is shown below for both MS Windows and Unix platforms.

For Windows

Here is the oblixpppcatalog.lst entry for MS Windows:

2e22c064723e4030a05b437e059fe4d6_1_postaction;
exec;uid;c:\j2sdk1.4.1_01\bin\java.exe; —classpath C:\ana\samples\bin Logger;;

- actionName—2e22c064723e4030a05b437e059fe4d6_1_postaction

  Here are the parts of the actionName for this example:

  - WORKFLOW-DEFINITION_ID

    2e22c064723e4030a05b437e059fe4d6

  - STEP-DEFINITION_NUMBER

    1

  - PPPTYPE

    postaction

- actionType—exec
- netpointparam1—uid
- path—/usr/local/bin/java

functionname— –classpath C:\ana\samples\bin Logger

Note: The same syntax applies for an entry pertaining to a pre-processing action. The single difference is that the PPPTYPE is preaction.

For Unix

Here is the oblixpppcatalog.lst entry for Unix

2e22c064723e4030a05b437e3059fe4d6_1_postaction;exec;uid;/usr/local/bin/java; -LD_LIBRARY_PATH/opt/ana/sample/bin Logger;;

actionName—2e22c064723e4030a05b437e3059fe4d6_1_postaction

actionType—exec

netpointparam1—uid

path—/usr/local/bin/java

functionname— –LD_LIBRARY_PATH/opt/ana/sample/bin Logger

Interaction Methods

Get

Operation	User Identity Key Name	Description of Data
GET	WfHandler	This is the callback URL expected by the asynchResumeWorkflowProcess function in IdentityXML. This URL will be of the form: http://www.customer.com/identity/oblix/apps/asynch/bin/asynch.cgi
GET	WfSubflow	A list of one or more of the subflows belonging to the current workflow.
GET	WfInstance.<attribute name>	A list of one or more values for the named attribute belonging to the current workflow. For example WfInstance.obtargetdn refers to the value for the target DN, as stored in the obtargetdn attribute. See the full list of WfInstance attributes in the following table.
GET	WfStepInstance.<attribute name>	A list of one or more values for the named attribute belonging to the current step of the current workflow. For example WfStepInstance.obactordn refers to the value for the uid of the person processing the current step, as stored in the obactordn attribute.
GET	WfAttribute.<attribute name>	A list of one or more values for the named Workflow attribute. Refers to the configured workflow attribute for the step.
GET	WfSubflow.<subflowid>.<attribute name>	A list of one or more values for the named attribute under the named subflow ID for the current Workflow, where *attribute name* is any of the WfInstance attributes pertaining to the subflowid. For example, WfSubflow.63f004504f83455b924133acd0ef87. obtargetdn refers to the target DN of the subflow whose ID is 63f004504f83455b924133acd0ef87 and which is triggered from the current workflow instance.

Set

Operation	User Identity Key Name	Description of Data
SET	WfAttribute. <attribute name>	Any of the configured workflow attributes for the step.
SET	WfInstance. Obwf supplementalval	Set the approval status for all subflows. Applies only to steps that have multiple workflows. Possible values to be set are: rejected approved

Receive

XML can be received just as for pre- and post-processing events, but only in the EventXML format.

Send

XML can be sent just as for pre- and post-processing events, but only in the EventXML format.

SetResultString

A string returned with this method will be displayed by the COREid application.

Tables of Workflow Attributes

The following table summarizes WfInstance attributes:

Attribute Name	Meaning
obactionindicator	For internal use.
obactorcomment	Comments entered during workflow processing.
obapp	Application to which the workflow belongs (for example userservcenter for User Manager).
obattr	For internal use.
obcertid	Certificate id (used by certificate workflows).
obclass	Object classes of the target entry.
obcurrentdn	The dn of the user who initiated the workflow.
obcurrentstep	The dn of the current step being processed.
obdatecreated	Integer date when the workflow instance was created.

Attribute Name	Meaning
obdateprocessed	Integer date when the workflow instance was last processed.
obhostname	Hostname of the machine from where the workflow was initiated.
obkey	For internal use.
oblockedby	The dn of the user who locked the workflow ticket.
obparentstep	Applicable to subflows. dn of the parent workflow step instance that triggered the subflow.
obparentworkflow	Applicable to subflows. The dn of the parent workflow instance.
obport	Port number of the machine from which the workflow was initiated.
obtargetdn	The dn of the target user entry.
obtriggeredworkflow	For internal use. Maintains number of unfinished triggered subflows.
obver	NetPoint version.
obwfinstanceid	Instance identifier.
obwfstatus	Status of workflow.

obwfsupplementalval	For NetPoint 6.x, this indicates a single approval status of all triggered subflows. Valid values:
	• approved
	• rejected
	Example:
	data->Get("WfInstance.obwfsupplementalval");
	For NetPoint 5.2, the attribute is subflowoutcome. Example:
	data->Get("subflowoutcome");

obwftypename	Display name of type of workflow.
obworkflowdn	The dn of the workflow definition.
obworkflowname	Name of workflow definition.
obworkflowtype	Type of workflow, such as create user, change attribute, and so on.

The following table summarizes WfStepInstance attributes:

Attribute Name	Meaning
obactionname	Step action (for example *initiate*, *request*, and so on.)
obactionreturncode	For internal use..
obactorcomment	Status message of step processing.
obactordn	The dn of the user processing the current step.
obapp	Application to which the workflow belongs (for example userservcenter for User Manager).
obdatecreated	Integer date when the workflow step instance was created.
obdateprocessed	Integer date when the workflow step instance was last processed.
obentrycondition	Not used.
obexitcondition	Not used.
oblockedby	The dn of the user who locked the workflow ticket.
oboptionalattribute	List of optional attributes configured for the step. The attribute list is comma delimited.
obparticipant	Not used.
obprovisionedattribute	List of attributes for which subflows are configured in the step. The attribute list is comma delimited.
obrequiredattribute	List of required attributes configured for the step. The attribute list is delimited with comma.
obretrycount	Number of times the step has been retried. Applicable only if retry status is returned from workflow event handler.
obretrydone	Boolean value that indicates if the retry is completed or not.
obtriggeredworkflow	Not used.
obver	NetPoint version.
obwfstatus	Status of step instance processing.
obwfstepinstid	Step instance identifier.
obworkflowstepdn	The dn of the workflow step definition.

Attribute Name	Meaning
obattr	Attribute name.
obattrtype	Attribute type.
obattrvals	Attribute values.
obver	NetPoint version.
obwfattrdefval	Not used.
obwfattrflags	Not used.

Return Values

The action *must* return one of four response values to the workflow engine:

- **STATUS_PPP_OK**—This value tells the workflow engine that the action has completed, and it may continue to the next step.
- **STATUS_PPP_ABORT**—This value tells the workflow engine an error has occurred. The workflow engine tells workflow participants for the current step in the workflow that it has failed, and uses its internal logic to handle the error.
- **STATUS_PPP_WF_ASYNC**—This value tells the workflow engine to put itself into a pending state, waiting for some external action to complete. Recover from this state by sending an asynchResumeWorkflowProcess command. The URL to which the command should be sent is requested using the parameter name wfhandler. This command and the process for using it, are described in the "IdentityXML Functions and Parameters" chapter.
- **STATUS_PPP_WF_RETRY**—This value tells the workflow engine that the step did not complete, most likely due to entry of invalid data. The user will need to try the step again, providing correct data. The current retry count is maintained in the directory entry for the workflow step instance. You can request the current retry count using the attribute name obretrycount.

Failure to formally return a response value may cause unpredictable behavior in the application, depending upon the default return value that the server operating system will supply instead.

Password Management Events

As part of creating a password policy, you may set a flag allowing “Externally specified validation rules.” If this flag is set on, then COREid checks the Catalog for actions to be used in place of its standard Password Management.

The event related to lost password management functionality is setChangedPassword and the application name is lost_pwd_mgmt. The sample application name, event name, and action is lost_pwd_mgmt_setChangedPassword_pre. Note that this is not the standard UserServCenter application naming convention.

For more information:

- “Catalog Entry” on page 213
- “Interaction Methods” on page 214
- “Return Values” on page 214

Catalog Entry

Under Password Management, only one event is possible, Password Validation. The action name therefore has the fixed value PWMGMT_PasswordValidation. Also, because no pre- or post-processing is supported, the name does not include the pre or post indicator that other actions use.

Field Name	Description
actionName (for Password Management)	Required. Provide this information in the form PWMGMT_PasswordValidation.
(other fields)	See the descriptions in “Configuration File (Catalog)” on page 179.

Here is an example of a password validation event entry in the Catalog:

```
PWMGMT_PasswordValidation;exec;;..\..\..\unsupported\ppp\ppp_exec\ppp_exec.exe;
;
```

This example calls ppp_exec.exe as an EXEC function to do password validation. This registers the stand-alone program ppp_exec.exe to perform password validation when a user attempts to change their password. The following must be true for the action to be invoked:

- The action must be configured in the catalog and deployed on the COREid System.
- A Master Identity Administrator must configure a password policy whose External Validation flag is turned on.
- The password policy must be enabled.

The domain of the password policy must contain the user's identity.

Interaction Methods

Get

Operation	User Identity Key Name	Description of Data
GET	Password	This is the password value entered by the user, to be validated. It is a two-member array, NULL terminated.
GET	PasswordPolicyDomain	The domain defined for the NetPoint password policy that applies to the user.
GET	PasswordPolicyFilter	The filter defined for the NetPoint password policy that applies to the user.

Set

This method is not supported for the Password Management event.

Receive

XML can be received just as for pre- and post-processing events, but only in the EventXML format.

Send

XML can be sent just as for pre- and post-processing events, but only in the EventXML format.

SetResultString

A string returned with this method will be displayed by the COREid application.

Return Values

The action must return one of two values:

- **STATUS_PPP_OK**—Indicates that the password conforms to the rules and can be changed to the indicated value.
- **STATUS_PPP_ABORT**—Indicates that it does not. The change can not be made.

Failure to formally return a response value will cause unpredictable behavior in the application, depending upon the default return value that the server operating system will supply instead.

Encryption Events

Whenever an encryption event occurs, COREid checks the Catalog for an encryption action. If one is present, then the process defined within it is used instead of COREid's default method.

If you make this change, NetPoint assigns all responsibility for the encryption to your action. Be sure that the encrypt and decrypt methods you use are the inverse of each other:

- "Catalog Entry" on page 215
- "Interaction Methods" on page 216
- "Response Values" on page 216

Catalog Entry

Catalog entries for encryption use the same format as pre- and post-processing events, with one difference, the actionName. Under Encryption, only two events are possible. Also, because no pre- or post-processing is supported, the name does not include the pre or post indicator that other actions use.

In table format:

Field Name	Description
`actionName` (for Encryption)	Required. Provide this information in the form APPNAME_EVENTNAME. Note this has only two parts, separated by underscores. APPNAME is the NetPoint application name, in this case Encryption, entered as ENCRYPTION. Under encryption, there are only two valid events, actually the type of information to be encrypted. These are the Cookie Encryption Key or the Challenge Response Encryption Key. Acceptable values for EVENTNAME are therefore cookieEncryptionKey or CPResponseEncryptionKey repectively.
(other fields)	See the descriptions in "Configuration File (Catalog)" on page 179.

Here is an example of an Encryption event entry in the Catalog.

```
ENCRYPTION_CPResponseEncryptionKey;lib;;
../../../unsupported/ppp/ppp_dll/ppp_dll.dll;
ProcessCPResponseEncryption;;
```

This example calls the `ProcessCPResponseEncryption` function in `ppp_dll.dll` to encrypt the challenge response key.

Interaction Methods

Get

Operation	User Identity Key Name	Description of Data
GET	OPERATION	This operation returns one of two values. ENCRYPT—Encrypt the data. DECRYPT—Decrypt the data.
GET	INPUTSTR	Returns user entered information in a two-member array, NULL terminated.

Set

Operation	User Identity Key Name	Description of Data
SET	OUTPUTSTR	The information to be returned to NetPoint. You must provide a NULL termination.

Receive

XML can be received just as for pre- and post-processing events, but only in the EventXML format.

Send

XML can be sent just as for pre- and post-processing events, but only in the EventXML format.

SetResultString

A string returned with this method will be displayed by the COREid application.

Response Values

The event must return one of two values:

- **STATUS_PPP_OK**—The event sends this value to indicate that encryption has completed satisfactorily.
- **STATUS_PPP_ABORT**—The event sends this value to indicate that encryption did not complete satisfactorily.

Note: Because encryption is essential to NetPoint's operation, any response other than STATUS_PPP_OK will cause the COREid instance to stop, and generate a NetPoint bug report.

The API

This section provides additional information for the developer on how to use the API:

Note: Do *not* use blank spaces in the names of any file in an Identity Event API project.

More on LIB Actions

You implement a LIB action as a callable function (with C language calling conventions), that resides within a dynamic shared object (DSO) library. The DSO must be native to the platform on which NetPoint is running. For example on NT, it must be a .dll; on Solaris UNIX it is must be an .so.

Note: When developing a LIB plug-in, global data must be implemented in a thread-safe manner.

LIB actions are executed in the address space of the NetPoint server process. It is critical that LIB actions be thoroughly tested before being deployed, as there is a class of programming errors (such as divide-by-zero errors) that cannot be caught by NetPoint and can cause the server to crash, or to exhibit other unstable behavior.

LIB actions communicate with the NetPoint application by calling API functions directly, passing the appropriate parameters, as described in "Connecting Events to Actions" on page 172.

More on MANAGEDLIB Actions

MANAGEDLIB actions are methods on a class. You implement a MANAGEDLIB action as a method on the EventAPI class, which is defined in the plug-in. The DSO is an assembly or a dll.

Note: When developing a MANAGEDLIB plug-in, member variables of class EventAPI must be accessed in a thread-safe manner.

MANAGEDLIB actions are executed in the address space of the NetPoint server process. It is critical that MANAGEDLIB actions be thoroughly tested before being deployed. However, any exception generated by the managed plug-in will be caught by the COREid Server, logged, and a bug report page will be generated.

MANAGEDLIB actions communicate with the NetPoint application by calling API functions directly, passing the appropriate parameters, as described in "Connecting Events to Actions" on page 172.

When compiling an EventAPI PPP Plug-In in VB.NET, ensure that:

a) Your VB Class is named "EventAPI"

b) You are not using namespaces in your code

c) You blank out the "Root Namespace" in the properties settings of your project in Visual Studio.NET with your VB project open:

   Go to the Project > Properties > Common Properties / General page > Root Namespace > remove the value > click OK.

The COREid Server will *not* load your dll if you fail to perform any of the items above.

Note: Any managed library to be used for PPP events must have the EventAPI class declared at the global namespace level. That is, it must be declared within no namespace. For a C# library, this means simply removing the 'namespace' directive from the source code. For a VB.Net library, remove the 'Default Namespace' option from the project.

More on EXEC Actions

You implement an EXEC action as a standalone executable. The action receives two kinds of input: command-line parameters that you specify in the catalog and XML data from the NetPoint program whose event is causing the action to run. The parameters are received in the ARGV[] array passed into the main() function (assuming a C/C++ programming environment). The XML data is available as a stream on the executable's standard input stream, STDIN.

Content of the XML data depends on whether the event is a pre-processing or post-processing event. For a pre-processing event, the data describes the request and is given in EventXML format. For a post-processing event, the data represents the result of processing the request and is given in PresentationXML format. (PresentationXML is the XML that would normally be combined with an XSL

stylesheet and transformed into the HTML ultimately seen by the user's browser.) The action is expected to perform its task and optionally write modified XML, in the same format as was received, back to the executable's standard output STDOUT. If information is returned on STDOUT, NetPoint receives it and generates the HTML based on the new data.

Note: A complete discussion of the process that COREid follows to logically combine XML data and XSL style sheets to create its HTML presentation is outside the scope of this manual. See the chapter on PresentationXML in the *NetPoint 7.0 Customization Guide*.

Use of STDIN and STDOUT gives developers the ability to code their EXEC actions in *any* language that supports these data streams. This includes Java, C, C++, PERL, Python, UNIX shells, and .NET languages such as C#, MC++, and VisualBasic. If it needs to access the data to perform its task, an EXEC action may invoke any XML parser to interpret the XML. COREid does *not* provide a built-in parser. The EXEC action *must* maintain the validity of the XML. Otherwise the Oblix-provided or custom XSL stylesheets that may be applied further downstream before presentation to the user may not produce the expected results. Within this constraint, the EXEC action may perform any processing needed:

- It might parse the NetPoint data and take action based on the result.
- It might filter the NetPoint data by replacing some or all of it in the output stream with different data, taking care to maintain compliance with the NetPoint data's XML schema.
- It might allow the NetPoint data to pass through untouched, but kick off another business process somewhere else on the network. For example, an action handling the workflowActivateSave event in User Manager can maintain a count of users that have been activated. When the count reaches a certain threshold, the action can trigger a backup or replication procedure to limit the risk of data loss. Or it can send email to an IT manager, who might want to investigate why so many users have suddenly been activated.

Returning Error Messages From an EXEC Call

There are three interfaces you can use to return error messages from an EXEC call. The three common interfaces are:

- EXEC - WF
- EXEC - PRE
- EXEC - POST

Returning Error Messages Using EXEC - WF

To send error message back to COREid, specify the 'ObResultString' in the XML along with the message to be displayed. The message is displayed in the confirmation page after the user processes a ticket.

The XML MUST be constructed as follows:

```
<ObEventParams>
<ObParam name="ObResultString">
<ObValue>The value of the result string goes here...</ObValue>
</ObParam>
<ObParamList name="WfAttribute">
<ObParam name="cn">
<ObValue>New value(s) go here...</ObValue>
</ObParam>
<ObParam name="sn">
<ObValue>New value(s) go here...</ObValue>
<ObValue>New value(s) go here...</ObValue>
</ObParam>
</ObParamList>
<ObParamList name="WfInstance">
<!--
NOTE: This is the only parameter that can be changed. The parameter is used to
set the outcome of a sub-flow. It will be displayed as the 'Outcome' value in the
'Subflow Approval' step. By default, COREid sets this parameter to 'approved' or
'rejected' in an 'Approval' step.
-->
<ObParam name="obwfsupplementalval">
<ObValue> New value(s) go here...</ObValue>
</ObParam>
</ObParamList>
</ObEventParams>
```

Returning Error Messages Using EXEC - PRE

For a PRE event, the XML is constructed in much the same way as in a WF event. The DOM is the same. The difference is in the parameter names. The parameters usually begin with ObRequest, followed by the parameter you wish to change. To set the result string for a PRE event, the executable must return 'STATUS_PPP_ABORT'.

```
<ObEventParams>
```

```
<ObParam name="ObRequest.uid">
<ObValue>cn=Thomas Remahl,o=Company,c=US</ObValue>
</ObParam>

<ObParam name="ObResultString">
<ObValue>Always viewing: cn=Thomas Remahl,o=Company,c=US</ObValue>
</ObParam>
</ObEventParams
```

Returning Error Messages Using EXEC - POST

In the case of a POST event, the XML must conform to the event that is associated with the plug-in. To set the result string for a POST event, you usually embed an <ObTextMessage> element in the XML returned to COREid. Whether the returned string is shown or not depends on the associated stylesheet.

For example, the output of the 'view' event below shows the returned string "Hello, World!". As already noted, the stylesheet determines if the element is applied. In the following example, the element is 'usc_profile.xsl'.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet href="../../../lang/en-us/style0/usc_profile.xsl" type="text/
xsl"?>
<Oblix xmlns:oblix="http://www.oblix.com/" xmlns="http://www.oblix.com/"
oblang="en-us">
<ObProfile>
<ObTextMessage>
Hello, World!
</ObTextMessage>
<ObPanel obname="defaultPanel" obpanelId="20040401T22135679142"
obpanelClass="gensiteorgperson">
<ObAttribute obattrName="genUserID">
<ObDisplay obdisplayName="UID" obdisplayType="textS" obsemanticType="ObSLogin"
obname="genUserID" obmode="view" obcanRequest="false" obrequired="false">
<ObTextS>
<ObValue>Admin</ObValue>
</ObTextS>
</ObDisplay>
</ObAttribute>
<ObAttribute obattrName="sn">
<ObDisplay obdisplayName="Last Name:" obdisplayType="textS" obname="sn"
obmode="view" obcanRequest="false" obrequired="false">
<ObTextS>
```

```
<ObValue>ÝdmÔn</ObValue>
</ObTextS>
</ObDisplay>
</ObAttribute>
<ObAttribute obattrName="cn">
<ObDisplay obdisplayName="Name" obdisplayType="textS" obname="cn" obmode="view"
obcanRequest="false" obrequired="false">
<ObTextS>
<ObValue>Master ÝdmÔn</ObValue>
</ObTextS>
</ObDisplay>
</ObAttribute>
</ObPanel>
<ObPanel obname="miisPanel" obpanelId="20040406T10492776123"
obpanelClass="gensiteorgperson">
<ObAttribute obattrName="cn.person.miis">
<ObDisplay obdisplayName="MIIS Name" obdisplayType="textS"
obname="cn.person.miis" obmode="view" obcanRequest="false" obrequired="false">
<ObTextS></ObTextS>
</ObDisplay>
</ObAttribute>
<ObAttribute obattrName="userSMIMECertificate.person.miis">
<ObDisplay obdisplayName="MIIS Password" obdisplayType="password"
obsemanticType="ObSPassword" obname="userSMIMECertificate.person.miis"
obmode="view" obcanRequest="false" obrequired="false">
<ObPassword oboldpsw="false"></ObPassword>
</ObDisplay>
</ObAttribute>
</ObPanel>
<ObHeaderPanel>
<ObAttribute obattrName="cn">
<ObDisplay obdisplayName="Name" obdisplayType="textS" obname="cn" obmode="view"
obcanRequest="false" obrequired="false">
<ObTextS>
<ObValue>Master ÝdmÔn</ObValue>
</ObTextS>
</ObDisplay>
</ObAttribute>
</ObHeaderPanel>
<ObRequestInfo>210498888</ObRequestInfo>
```

```
<ObScripts>
<ObScript obname="../../../lang/en-us/msgctlg.js"></ObScript>
<ObScript obname="../../../lang/shared/i18n.js"></ObScript>
<ObScript obname="../../../lang/shared/nsiesetup.js"></ObScript>
<ObScript obname="../../../lang/shared/misc.js"></ObScript>
<ObScript obname="../../../lang/shared/miscsc.js"></ObScript>
<ObScript obname="../../../lang/shared/horizontalprofile.js"></ObScript>
<ObScript obname="../../../lang/shared/userservcenter.js"></ObScript>
</ObScripts>
<ObForm obname="profileForm" obmethod="post"
obaction="userservcenter.cgi?tab_id=Employees&amp;uid=cn%3DMaster%20%C5dm%EFn%2
Co%3DCompany%2Cc%3DUS">
<ObInput obtype="hidden" obname="program" obvalue="view"></ObInput>
<ObInput obtype="hidden" obname="visiblePanel"></ObInput>
</ObForm>
<ObDisplay obdisplayName="ObTextMessage" obdisplayType="textS"
obname="ObTextMessage" obmode="view" obcanRequest="false" obrequired="false">
<ObTextS>
<ObTextMessage></ObTextMessage>
</ObTextS>
</ObDisplay>
<ObTextMessage></ObTextMessage>
<ObSelectorInfoForm>
<ObForm obname=""></ObForm>
</ObSelectorInfoForm>
<ObButton obaction="initiateDeactivateUser"></ObButton>
<ObButton obaction="userreactivate"></ObButton>
<ObButton obaction="wfTicketDelete"></ObButton>
<ObButton obaction="userModify" obimageUrl="NAVmodify" obmouseOver="Modify this
profile." obhref="../../userservcenter/bin/
userservcenter.cgi?program=modify&amp;tab_id=Employees&amp;uid=cn%3DMaster%20%C
5dm%EFn%2Co%3DCompany%2Cc%3DUS"></ObButton>
<ObStatus>0</ObStatus>
</ObProfile>
<ObNavbar obbgcolor="#669966">
<ObMisc>
<ObButton obaction="T1help" obimageUrl="T1help" obmouseOver="View Online Help"
obhref="javascript:ObHelp('../../help/bin/
help.cgi?program=helpProgram&amp;helpAppContext=userservcenter&amp;helpEventCon
text=view&amp;helpTOCContext=application');"></ObButton>
```

```
<ObButton obaction="T1about" obimageUrl="T1about" obmouseOver="Product
Information and Feedback"
obhref="userservcenter.cgi?program=aboutOblix&amp;tab_id=Employees"></ObButton>

<ObButton obaction="T1logout" obimageUrl="T1logout" obmouseOver="Logout"
obhref="userservcenter.cgi?program=commonLogout&amp;sessionUid=20040407T1438028
5745"></ObButton>

</ObMisc>

<ObApps>

<ObApplication>

<ObButton obaction="userservcenter_application_info"
obimageUrl="T1TABusermanager" obmouseOver="User Manager" obhref="../../
userservcenter/bin/userservcenter.cgi" obanchorText="User Manager"></ObButton>

<ObTitle>

<ObButton obaction="T1TABusermanager"></ObButton>

</ObTitle>

<ObFunctions>

<ObButton obaction="MyProfile" obimageUrl="FTABmyidentity2" obmouseOver="View my
profile."
obhref="userservcenter.cgi?program=view&amp;uid=cn%3DMaster%20%C5dm%EFn%2Co%3DC
ompany%2Cc%3DUS&amp;tab_id=Employees"></ObButton>

<ObButton obaction="Report" obimageUrl="FTABreports" obmouseOver="Report
Functions"
obhref="userservcenter.cgi?program=mainReports&amp;appName=userservcenter&amp;t
ab_id=Employees"></ObButton>

<ObReportFunctions>

<ObButton obaction="generateReport" obimageUrl="2FTABgeneratereport"
obmouseOver="Generate a report" obhref="javascript:QueryBuilder('../../
querybuilder/bin/
querybuilder.cgi?program=modifyFilter&amp;tab_id=Employees&amp;appName=userserv
center&amp;uid=cn%'+'3DMaster%'+'20%'+'C5dm%'+'EFn%'+'2Co%'+'3DCompany%'+'2Cc%'
+'3DUS&amp;advModeDisable=true&amp;slapTab=true','','..%'+'2F..%'+'2Fuserservce
nter%'+'2Fbin%'+'2Fuserservcenter.cgi%'+'3Fprogram%'+'3DshowReportsResults%'+'2
6appName%'+'3Duserservcenter%'+'26tab_id%'+'3DEmployees%'+'26fromQB%'+'3Dtrue',
'')"></ObButton>

<ObButton obaction="viewPredefinedReports"
obimageUrl="2FTABviewpredefinedreports" obmouseOver="View predefined reports"
obhref="userservcenter.cgi?program=predefinedReports&amp;appName=userservcenter
&amp;tab_id=Employees"></ObButton>

</ObReportFunctions>

<ObButton obaction="wfCreateProfile" obimageUrl="FTABcreateuseridentity"
obmouseOver="Create New User"
obhref="userservcenter.cgi?program=workflowCreateProfile&amp;tab_id=Employees">
</ObButton>
```

```
<ObButton obaction="wfDeactivateProfile" obimageUrl="FTABdeactivateuseridentity"
obmouseOver="Search on Deactivated Persons."
obhref="userservcenter.cgi?program=workflowDeactivatedUserSearchResults&amp;tab
_id=Employees"></ObButton>

<ObButton obaction="adminProxy" obimageUrl="FTABsubstituterights"
obmouseOver="Configure Proxy Administration"
obhref="userservcenter.cgi?program=proxyAdmin&amp;tab_id=Employees"></ObButton>

<ObButton obaction="Workflow" obimageUrl="FTABrequests" obmouseOver="Workflow
Functions"
obhref="userservcenter.cgi?program=workflowMain&amp;tab_id=Employees"></
ObButton>

<ObWorkflowFunctions>

<ObButton obaction="wfIncomingRequest" obimageUrl="2FTABincomingrequests"
obmouseOver="Incoming Request" obhref="../../userservcenter/bin/
userservcenter.cgi?program=workflowTicketSearchForm&amp;tab_id=Employees&amp;re
questType=incomingRequests"></ObButton>

<ObButton obaction="wfOutgoingRequest" obimageUrl="2FTABoutgoingrequests"
obmouseOver="Outgoing Request" obhref="../../userservcenter/bin/
userservcenter.cgi?program=workflowTicketSearchForm&amp;tab_id=Employees&amp;re
questType=outgoingRequests"></ObButton>

<ObButton obaction="wfMonitor" obimageUrl="2FTABmonitorrequests"
obmouseOver="Requests Monitor" obhref="../../userservcenter/bin/
userservcenter.cgi?program=workflowMonitorSearchForm&amp;tab_id=Employees"></
ObButton>

</ObWorkflowFunctions>

<ObButton obaction="Admin" obimageUrl="FTABconfiguration"
obmouseOver="Administrative Functions"
obhref="userservcenter.cgi?program=administrationMain&amp;tab_id=Employees"></
ObButton>

<ObAdminFunctions>

<ObButton obaction="adminAccessControl" obimageUrl="2FTABattraccesscontrol"
obmouseOver="Configure Attribute Access Control"
obhref="javascript:DetectPluginForApplets('../../userservcenter/bin/
userservcenter.cgi?program=mainAccessAdmin&amp;tab_id=Employees')"></ObButton>

<ObButton obaction="adminDelegate" obimageUrl="2FTABdelegateadmin"
obmouseOver="Configure Delegated Administration"
obhref="javascript:DetectPluginForApplets('../../userservcenter/bin/
userservcenter.cgi?program=mainDelegateAdmin&amp;tab_id=Employees')"></
ObButton>

<ObButton obaction="adminWorkflowDef" obimageUrl="2FTABworkflowdefinition"
obmouseOver="Configure Workflow Definition"
obhref="javascript:DetectPluginForApplets('../../userservcenter/bin/
userservcenter.cgi?program=mainWorkflowAdmin&amp;tab_id=Employees')"></
ObButton>
```

```
<ObButton obaction="adminSetSearchbase" obimageUrl="2FTABsetsearchbase"
obmouseOver="Configure Localized Access"
obhref="javascript:DetectPluginForApplets('../../userservcenter/bin/
userservcenter.cgi?program=mainSetSearchbase&amp;tab_id=Employees')"></
ObButton>
</ObAdminFunctions>
</ObFunctions>
</ObApplication>
<ObApplication>
<ObButton obaction="groupservcenter_application_info"
obimageUrl="T1TABgroupmanager" obmouseOver="Group Manager" obhref="../../
groupservcenter/bin/groupservcenter.cgi" obanchorText="Group Manager"></
ObButton>
<ObTitle></ObTitle>
<ObFunctions></ObFunctions>
</ObApplication>
<ObApplication>
<ObButton obaction="objservcenter_application_info" obimageUrl="T1TABorgmanager"
obmouseOver="Org. Manager" obhref="../../objservcenter/bin/objservcenter.cgi"
obanchorText="Org. Manager"></ObButton>
<ObTitle></ObTitle>
<ObTabs></ObTabs>
<ObFunctions></ObFunctions>
</ObApplication>
<ObApplication>
<ObButton obaction="corpdir_application_info"></ObButton>
<ObTitle></ObTitle>
<ObTabs></ObTabs>
<ObFunctions></ObFunctions>
</ObApplication>
<ObApplication>
<ObButton obaction="dashline" obmouseOver="-----------------------------------"
obhref="userservcenter.cgi?"></ObButton>
</ObApplication>
<ObApplication>
<ObButton obaction="front_page_admin_application_info"
obimageUrl="T1TABidentityadmin" obmouseOver="COREid System Console" obhref="../
../admin/bin/front_page_admin.cgi" obanchorText="COREid System Console"></
ObButton>
</ObApplication>
</ObApps>
<ObScripts>
```

```
<ObScript obname="../../../lang/en-us/msgctlg.js"></ObScript>
<ObScript obname="../../../lang/shared/i18n.js"></ObScript>
<ObScript obname="../../../lang/shared/misc.js"></ObScript>
<ObScript obname="../../../lang/shared/helpcommon.js"></ObScript>
<ObScript obname="../../../lang/shared/wf_qs.js"></ObScript>
</ObScripts>
<ObStatus>0</ObStatus>
<ObUserName>Master ÝdmÔn</ObUserName>
</ObNavbar>
<ObSearchForm>
<ObSearchRow>
<ObDisplay obdisplayName="" obdisplayType="select" obname="STy1" obmode="modify"
obrequired="true" obcardinality="singleValued" obcanRequest="false">
<ObSelect obmultiple="false">
<ObChoice obdisplayName="Last Name:" obselected="false">sn</ObChoice>
<ObChoice obdisplayName="MIIS Name" obselected="false">cn.person.miis</ObChoice>
<ObChoice obdisplayName="Name" obselected="true">cn</ObChoice>
<ObChoice obdisplayName="UID" obselected="false">genUserID</ObChoice>
</ObSelect>
</ObDisplay>
<ObDisplay obdisplayName="" obdisplayType="select" obname="SLk1" obmode="modify"
obrequired="false" obcardinality="singleValued" obcanRequest="false">
<ObSelect obmultiple="false">
<ObChoice obdisplayName="That Contains" obselected="false">OOS</ObChoice>
<ObChoice obdisplayName="Contains In Order" obselected="false">OSM</ObChoice>
<ObChoice obdisplayName="=" obselected="false">OEM</ObChoice>
<ObChoice obdisplayName="&lt;=" obselected="false">OLE</ObChoice>
<ObChoice obdisplayName="&gt;=" obselected="false">OGE</ObChoice>
<ObChoice obdisplayName="That Begins With" obselected="false">OBW</ObChoice>
<ObChoice obdisplayName="That Ends With" obselected="false">OEW</ObChoice>
<ObChoice obdisplayName="That Sounds Like" obselected="false">OSL</ObChoice>
<ObChoice obdisplayName="!=" obselected="false">ONE</ObChoice>
</ObSelect>
</ObDisplay>
<ObDisplay obdisplayName="" obdisplayType="textS" obname="SSt1" obmode="modify"
obrequired="false" obcardinality="singleValued" obcanRequest="false">
<ObDisplayProperties>
<ObDisplayProperty obname="onKeyDown"
obvalue="javascript:checkSearchKey(event,this)"></ObDisplayProperty>
</ObDisplayProperties>
```

```
<ObTextS oblength="19"></ObTextS>
</ObDisplay>
</ObSearchRow>
<ObAdvancedSearch obadvancedSearchOn="false">
<ObDisplay obdisplayName="" obdisplayType="radio" obname="showAllResults"
obmode="modify" obrequired="true" obcardinality="singleValued"
obcanRequest="false">
<ObRadio>
<ObChoice obdisplayName="All" obselected="false">true</ObChoice>
<ObChoice obdisplayName="" obselected="true">false</ObChoice>
</ObRadio>
</ObDisplay>
<ObDisplay obdisplayName="" obdisplayType="textS" obname="noOfRecords"
obmode="modify" obrequired="true" obcardinality="singleValued"
obcanRequest="false">
<ObTextS oblength="2">
<ObValue>8</ObValue>
</ObTextS>
</ObDisplay>
</ObAdvancedSearch>
<ObRequestInfo>210498888</ObRequestInfo>
<ObScripts>
<ObScript obname="../../../lang/en-us/msgctlg.js"></ObScript>
<ObScript obname="../../../lang/shared/i18n.js"></ObScript>
<ObScript obname="../../../lang/shared/misc.js"></ObScript>
</ObScripts>
<ObForm obname="searchForm" obmethod="post" obaction="userservcenter.cgi?">
<ObInput obtype="hidden" obname="program" obvalue="search"></ObInput>
<ObInput obtype="hidden" obname="tab_id" obvalue="Employees"></ObInput>
<ObInput obtype="hidden" obname="startFrom" obvalue="0"></ObInput>
<ObInput obtype="hidden" obname="getPrevRecords" obvalue="false"></ObInput>
<ObInput obtype="hidden" obname="noOfFields" obvalue="1"></ObInput>
<ObInput obtype="hidden" obname="displayFormat" obvalue="2"></ObInput>
<ObInput obtype="hidden" obname="advSearch" obvalue="false"></ObInput>
<ObInput obtype="hidden" obname="searchStringMinimumLength" obvalue="3"></
ObInput>
<ObInput obtype="hidden" obname="searchSameAttrAsOr" obvalue="false"></ObInput>
</ObForm>
<ObDisplay obdisplayName="ObTextMessage" obdisplayType="textS"
obname="ObTextMessage" obmode="modify" obrequired="false"
obcardinality="singleValued" obcanRequest="false">
```

```
<ObTextS>
<ObTextMessage></ObTextMessage>
</ObTextS>
</ObDisplay>
<ObSelectorInfoForm>
<ObForm obname=""></ObForm>
</ObSelectorInfoForm>
<ObButton obaction="searchGo" obimageUrl="SEARCHgo" obmouseOver="Start search."
obhref="javascript:validateSearchAndSubmit('search')"></ObButton>
<ObButton obaction="searchAdvance" obimageUrl="SEARCHadvanced"
obmouseOver="Advanced search."
obhref="javascript:validateSearchAndSubmit('moreFields')"></ObButton>
<ObButton obaction="searchLess"></ObButton>
<ObButton obaction="searchMore" obimageUrl="SEARCHmore" obmouseOver="Get more
fields." obhref="javascript:validateSearchAndSubmit('moreFields')"></ObButton>
<ObButton obaction="searchAll" obimageUrl="SEARCHall" obmouseOver="All fields."
obhref="javascript:validateSearchAndSubmit('allFields')"></ObButton>
<ObStatus>0</ObStatus>
</ObSearchForm>
<ObStatus>0</ObStatus>
</Oblix>
```

Development Environment

The Identity Event Plug-in API consists of a set of header files that you can use to build your LIB actions, source code examples for working with LIB, MANAGEDLIB, and EXEC actions, source code examples for creating an XML parser, and a default obpppcatalog.lst file with extensive examples of action configuration entries. On Windows platforms, an import library is also provided, which you will need to build your LIB actions. All of these files are bundled in the standard NetPoint COREid System installation; there is nothing else to install in order to develop actions.

For managed code, plug-in writers need to compile and link with pppInterface.dll, which contains the IPPPData interface. This assembly is located at

install_dir/oblix/include/managed/pppinterface.dll

This path will need to be referenced as a “Resolve #using Reference”, in Visual Studio, or through the /AI compiler option when compiling and linking the plug-in. At runtime, both the COREid Server and the plug-in will need to locate pppInterface.dll. For this reason, pppInterface.dll is installed in the Global Assembly Cache (GAC) during installation of the COREid Server. Alternatively, if plug-in writers wish to test their plug-in on a machine where COREid has not

been installed, pppInterface.dll can be privately deployed. This means placing the assembly in the plug-in's bin directory. It is important to compile and link with the same version of pppInterface.dll that will be used at runtime (either through the GAC or through private deployment). Otherwise, an exception may be thrown by the Common Language Runtime (CLR).

The files you need to be familiar with in order to develop custom actions are described in the following tables:

Library Files for LIB and EXEC Actions

Directory:

$COREid_install_Dir/oblix/lib

File Name	Description
`ppp.lib`	(Windows platforms only.) An export library you need to link your LIB action DLL to in order to resolve references to NetPoint-provided symbols.

Directory:

$COREid_install_Dir/oblix/include/ppp

File Name	Description
`obppp.h`	Defines the basic success/failure status return codes used by the API functions. Declares the ObActionFunc function signature that you must use to declare your LIB action functions.
`obpppdata.h`	Defines the ObPPPData C++ class that you must use to transmit data between NetPoint and your LIB action.
`obpppwf.h`	Defines constants used for developing actions that work with NetPoint's workflow functionality.

Library Files for MANAGEDLIB Actions

The pppInterface.dll is located as follows:

$COREid_install_dir\identity\oblix\include\managed\

This is the dll with which plug-ins compile and link.

File Name	Description
`pppInterface.dll`	(Windows platforms only.) The DSO for MANAGEDLIB actions.

LIB Action Example Files

These are examples only, *not* part of the product. See the \unsupported branch of the NetPoint directory tree. Directory:

COREid_install_dir/oblix/unsupported/ppp/ppp_dll

File Name	Description
libppp_dll.so	(Solaris UNIX platforms only.) This is a dynamic shared object (DSO) that is pre-built from source files present in this example directory. You provide a path to this DSO as part of the entry for a lib action in the oblixpppcatalog.lst file, specifying the name of one of the action functions within the DLL, as defined in pppdlltest.cpp. You must also include the path to the libppp_dll.so DSO in the shared library search path. The preferred method for doing this is to use the -L option of the ld command. Another way is to use the LD_LIBRARY_PATH environment variable, which can be set to give the run-time shared library loader (ld) an extra set of directories to look for when it searches for DSOs.
ppp_dll.dll	(Windows platforms only.) This is a dynamic link library (DLL) pre-built from source files in this example directory. You provide a path to this DLL as part of the entry for a lib action in the oblixpppcatalog.lst file, specifying the name of one of the action functions within the DLL, as defined in pppdlltest.cpp.
ppp_dll.sln	(Windows platforms only.) A Microsoft Visual C++ project file you can use to build ppp_dll.dll yourself.
pppdlltest.cpp	The C++ source file for the Oblix-provided LIB action examples.
ppputil.cpp	Provides: A C++ class that illustrates how to access the COREid System data available to actions through the API. The example simply writes the data out to the file system. A C function, MakePayload, that illustrates how to compose an XML SOAP message to request a group subscription for a user.
ppputil.h	Class and function declarations for ppputil.cpp.
nis_client.cpp	Provides a C++ class that implements an HTTP client capable of sending messages to NetPoint using WebPass. By combining this with the MakePayload function mentioned under ppputil.cpp, your action can use the IdentityXML or AccessXML interfaces to make requests of NetPoint. This example may be particularly useful for Cross Application Support. See page 234.
nis_client.h	Class and function declarations for nisclient.cpp.
makefile	(Solaris UNIX platforms only.) This is the UNIX make file used to create libppp_dll.s.

MANAGEDLIB Action Example Files

Directory:

COREid_install_dir\unsupported\ppp\dotnet\managedcplusplus\Release

File Name	Description
`managedcplusplus.cpp`	(Windows platforms only.) The MC++ source file for the Oblix-provided MANAGEDLIB action examples.
`managedcplusplus.h`	(Windows platforms only.) The header file for managedcplusplus.cpp. It defines a singleton class that contains methods specified as Identity Event API actions.
`managedcplusplus.sln`	(Windows platforms only.) A Microsoft Visual C++ managed code project file that you can use to build managedcplusplus.dll.
`managedcplusplus.vcproj`	(Windows platforms only.) A Microsoft Visual C++ managed code project file that contains the necessary configuration to build the project.
`managedcplusplus.dll`	The sample plug-in.
`pppfilewriter.cpp`	(Windows platforms only.) A utility class that receives COREid data and writes the data to a file.
`pppfilewriter.h`	(Windows platforms only.) The header file for pppfilewriter.cpp.

EXEC Action Example Files

Directory:

COREid_install_dir/oblix/unsupported/ppp/ppp_exec

File Name	Description
`ppp_exec_test.java`	This is the source for a JAVA version of an Oblix-provided EXEC action example for post-processing. You can refer to this program as an EXEC action in the oblixpppcatalog.lst file.
`ppp_exec.exe`	(Windows platforms only.) This is an NT executable, pre-built from the file pppexectest.cpp to make that example available to you. You can refer to this program as an EXEC action in the oblixpppcatalog.lst file.
`ppp_exec.sln`	(Windows platforms only.) A Microsoft Visual C++ project file you can use to build ppp_exec.exe yourself.
`pppexectest.cpp`	The C++ source file for an Oblix-provided EXEC action example.

File Name	Description
ppp_perl.pl	This is the source for a PERL version of an Oblix provided EXEC action example for post-processing. You can refer to this program as an EXEC action in the oblixpppcatalog.lst file.
ppp_string.cpp	A C++ class representing strings used by pppexectest.cpp.
ppp_string.h	Class declaration for ppp_string.cpp.
corpdir_view_pre.xml	A pre-formatted XML message for the example to send to NetPoint when invoked as a preprocessing step. See the pppexectest.cpp example.

Parser Example Files

Directory:

COREid_install_dir/oblix/unsupported/ppp/parser_test

File Name	Description
MyPPPActions.cpp	The C++ source file that builds a function called SAXParserPostActionTest, to be loaded as part of a DSO called MYPPPActions.dll. The file also provides Windows and UNIX examples of the Catalog entry that connects the action to the view post-processing event in the Profile page of the User Manager. The function replaces the phone numbers of corporate users with the pattern XXX-XXX-XXXX.
MyPPPActions.dll	This is the dynamic link library (DLL) pre-built from source files in this example directory.
MyPPPAction.sln	(Windows platforms only.) A Microsoft Visual C++ project file you can use to build MyPPPActions. dll yourself.
MySAXhandler.cpp	The C++ source file that builds the SAXhandler class of methods that does the actual interpretation of the XML. SAX stands for Simple API for XML.
MySAXhandler.hpp	The header file defining the methods belonging to the SAXhandler class.

Note: The examples are provided for illustrative purposes only. To emphasize that they are *not* part of the formal product, they are installed in the unsupported branch of the NetPoint directory tree.

Cross-Application Support

Standard workflows exist within specific applications, such as the User Manager and Group Manager, and their direct effects are limited to the application in which they exist. Situations may arise in which you want a workflow to make changes that affect more than one Manager application. An example is the need to create a new user and also subscribe that user to a Group.

This is accomplished by including an event in the workflow, which triggers an action that gets information from the workflow, and uses IdentityXML syntax to send a request to the other application to accomplish the task. The flow might be something like this:

- The event is invoked in the usual way as part of the workflow. NetPoint provides parameters, such as the user DN and group(s) to be subscribed to, to the corresponding action.
- The action combines this information into a subscribeUserToGroup IdentityXML request. The IdentityXML request requires a login id, password, and URL for the Group Manager. None of this will have been known to the creator of the workflow. The action will need to get it somehow. It could be coded into the action, extracted from a database, or provided by a file. For our example, we use a file named conf.txt.
- The information from NetPoint is combined with the information from the file to build the IdentityXML request, and the request is sent to the Group Manager URL. There, it is accepted and carried out, or denied.
- The status returned by IdentityXML is received by the action, interpreted and returned to the NetPoint application as either STATUS_PPP_OK or STATUS_PPP_ABORT.

The event entry in the Catalog to implement this might be the following:

```
63f004504f83455b924133acd0ef2e87_3_postaction;
lib;;../../../unsupported/ppp/ppp_dll/ppp_dll.dll;
NISClient;
```

It takes the same form as any other Workflow event. The behavior difference lies in the NISClient function, which performs all the duties described in the previous list. You will find the example code for the NISClient function in the file pppdlltest.cpp, with supporting methods in nis_client.cpp, both in the directory

COREid_install_dir/oblix/unsupported/ppp/ppp_dll

The example conf.txt file is located in:

COREid_install_dir/oblix/unsupported/ppp/ppp_dll

If you use it, you will need to change the content to match your situation, and move it to where the dll expects to find it:

COREid_install_dir/oblix/apps/common/bin

Note: There may be timing delays involved when you use a Cross Application plug-in. For example, if you are using replicated directories it will take time for information written to a first directory to be duplicated to a second. Your plug-in should allow for this time difference before trying to use data from the second directory.

Examples

A LIB Action Example—LogActivation

In this example, we will examine a C function that implements logging for both activation and deactivation of users in the COREid System. Notice that the same action function is registered in the Catalog for two different events. The event name is passed to the action, so it can differentiate between events that are handled in similar ways.

In the example, the log is written to the file system. A more sophisticated implementation might connect directly to a relational database to collect statistics like this for later processing by external enterprise applications. You should resist the urge to do too much in an action, however. Time spent in an action is time added to the of the HTTP request latency perceived by the user, in this case a Delegated Identity Administrator.

The following code implements this feature, packaged as a LIB action:

```
#include <ppp/obppp.h>
#include <ppp/obpppwf.h>
#include <ppp/obpppdata.h>

extern "C" {

/**
* LogActivation
* This action logs user activation and deactivation
* events.
* @param eventName The name of the event that
* triggered this action.
*              This example processes both activation and
*             deactivation, and uses this parameter to
*              tell the difference.
* @param data the data for this event.
*  (re: include/ppp/obpppdata.h)
* @return STATUS_PPP_OK or STATUS_PPP_ABORT
```

```
**/

unsigned int
LogActivation(const char *eventName, ObPPPData *data)
{
   // Event names (must match those used in catalog)
   const char *ACTIVATE_EVENT =
        "userservcenter_workflowActivateSave_pre";
   const char *DEACTIVATE_EVENT =
        "userservcenter_workflowDeactivateUserSave_pre";
   // open our file
   FILE *file = fopen("activation_log.txt", "a");
   // Determine whether action is being called to log
   // an activate or deactivate user event.
   bool activate;
   if (0 == strcmp(eventName, ACTIVATE_EVENT)) {
                activate = true;
   } else if (0 == strcmp(eventName, DEACTIVATE_EVENT)) {
                     activate = false;
                   } else {
      // error - can't process other events
                   data->SetResultString("PPP action misconfigured");
                   fclose(file);
                   return STATUS_PPP_ABORT;
                   }

   const char **uid = (const char **)data->Get("uid");
   if (NULL == *uid) {
                data->SetResultString("PPP action error");
                fclose(file);
                return STATUS_PPP_ABORT;
      data->SetResultString("PPP action error");
      fclose(file);
      return STATUS_PPP_ABORT;
   }
   // Write the log entry
   fprintf(file, "%s: %s\n",
                activate ? "activated" : "deactivated",
                *uid);
   fclose(file);
   return STATUS_PPP_OK;
}
```

For reference in the following description, here is how this action can be configured in oblixpppcatalog.lst on a UNIX system:

```
userservcenter_workflowActivateSave_pre;lib;;/var/opt/netpoint/plug-ins/
liblogactions.so;LogActivation;

userservcenter_workflowDeactivateUserSave_pre;lib;;/var/opt/netpoint/
plug-ins/liblogactions.so;LogActivation;
```

The `LogActivation` LIB action begins by including the Identity Event Plug-in API header files, as all LIB actions must do in order to have access to NetPoint data.

Notice that the `LogActivation` is declared within an extern C block to tell the C++ compiler that it is code written in C with external C linkage.

Next is the function signature for the action:

```
unsigned int
LogActivation(const char *eventName, ObPPPData *data)
```

This code declares `LogActivation` as a function with the same return type and parameter list as an ObActionFunc, as described in obppp.h. NetPoint requires that all LIB actions conform to this type.

LogActivation then declares constants for ACTIVATE_EVENT and DEACTIVATE_EVENT. The values of these constants reflect the events that this action will respond to, and must match the stylized event names used in the Catalog, as shown in the preceding code listing.

Next, a file is opened for append using fopen(). This is the log file for the example. It resides in the current working directory of NetPoint COREid System, which is the identity/oblix/apps/common/bin directory. In this example, there are just two possible types of entry in the log file:

- `activated: <user dn>`
- `deactivated: <user dn>`

LogActivation next inspects the name of the event for which it is being invoked, and sets an activate/deactivate flag. Then it looks up the DN of the user using the Get method of ObPPPData, to fetch the value of the uid parameter. The value of this parameter for this event is the directory DN of the user who is being activated or deactivated.

Note: Notice that the action demonstrates communicating an error to User Manager by setting the return status to STATUS_PPP_ABORT if it is called for an unexpected event, or if it fails to find the expected data in the ObPPPData object.

The action completes its task by writing its log message, closing the log file and returning the success status, STATUS_PPP_OK to User Manager.

An EXEC Action Example—AfterHours

This example implements an after-hours lockout function using a post-processing EXEC action. The intent is that a site may have a policy of disallowing certain types of activity during certain hours of the day, to allow a safe environment for backups and other system maintenance. This action might be one tool in the administrator's toolbox for enforcing such a policy.

Here is the source code for the AfterHours action:

```
#include <time.h>
#include <stdio.h>
#include <string.h>
#include <iostream.h>
#include <stdlib.h>
#include <ppp/obppp.h>
int main(int argc, char* argv[])
{
   // XML template for text message
   static const char *messageTemplate =
      "<?xml version=\"1.0\" encoding=\"ISO-8859-1\" ?>\n\
       <?xml-stylesheet href=\"../../common/ui/style0/ppp.xsl\"
            type=\"text/xsl\"?>\n \
       <Oblix xmlns=\"http://www.oblix.com/\">\n \
       <ObTextMessage>\n \
       %s\n \
       </ObTextMessage>\n \
       </Oblix>\n";

   static const char *message;

   if (argc > 1 && argv[1] != 0 && stricmp(argv[1], "pre")
                        == 0) {
      // PRE-processing requests are not supported
                return(STATUS_PPP_ABORT);
   } else {
      // POST-processing
      // Examine command-line for any EXEC arguments
                if (argc > 1 && argv[1] != 0) {
                   const long now = time(0);
                   struct tm* tmNow = localtime(&now);
                   int hrsNow = tmNow->tm_hour;
                   int minNow = tmNow->tm_min;
                   int hrsOff = atoi(argv[2]);
                   int minOff = atoi(argv[3]);
                   int hrsOn = atoi(argv[4]);
                   int minOn = atoi(argv[5]);
                   int timeOff = (60 * hrsOff) + minOff;
                   int timeOn = (60 * hrsOn) + minOn;
                   int timeNow = (60 * hrsNow) + minNow;
                   if (timeOn < timeOff) timeOn += (60*24);
                   if (timeOn != timeOff && timeNow >= timeOff &&
```

```
                timeNow < timeOn) {
            // Disallow the event; send ObTextMessage using
            // text in catalog
                      message = argv[1];
                    } else {
            // Allow the event. As a convenience, NetPoint
            // applications assume actions haven't modified
            // the data if they don't write to stdout. So
            // all you need to do here is return status.
                    return STATUS_PPP_OK;
                    }
                  } else {
         // No arguments. Output a default disablement message.
                      message = "This operation is disabled by NetPoint
POST-processing action.";
                  }
      // If we get here, we're replacing the data with
      // the ObTextMessage.
                fprintf(stdout, messageTemplate, message);
                fflush(stdout);
                return(STATUS_PPP_OK);
                }
}
```

Here is a sample Catalog entry to configure the AfterHours action on a Windows server.

```
userservcenter_view_post;exec;;C:\NetPoint\Identity\Actions\AfterHours.e
xe;"This Operation is unavailable outside business hours. Please contact
your Identity administrator for details." 21 30 06 00;
```

The first field associates the action with the User Manager view post-processing event. The second field is empty (no NetPoint parameters). The third field indicates that this is an EXEC action. The fourth field is the path to the executable that implements the AfterHours action. The remaining fields are EXEC action parameters, and they are supplied to the action as argv[1] through argv[5]. Notice that the text message parameter must be quoted because it contains spaces. The last four parameters indicate that the OFF hours are 21:30 (9.30pm) till 06:00 (6.00am).

Note: This illustrates the use of action parameters. Parameters are only available to EXEC actions, not LIB actions. A LIB action that implemented the after-hours lockout feature would have to look up its OFF and ON hours, and the text message to be displayed from an external source. Doing so would provide the opportunity for greater sophistication: just like a home-security time switch, an administrator may want more than one OFF period per day, or may want a different schedule on weekends. Knowing the requirements will help you to design your action interface, and help you decide whether a LIB or EXEC action is called for.

The AfterHours action begins by declaring a string containing the XML document that is used to return a text message to the browser if the event is currently disabled. Notice the %s embedded in the string. The string is used as a template to fprintf; the %s is a printf directive and is replaced by the actual message.

Next, the action rejects attempts to call it from a pre-processing event. It does not support pre-processing because it cannot usefully replace the XML result of the request until it has been generated, which is not until after pre-processing.

AfterHours then performs some time calculations. To do this, it checks the system time and extracts the current hour and minute, then converts it to minutes alone. It then examines its command-line arguments, and extracts the Catalog-supplied message, the OFF hours and minutes, and the ON hours and minutes, as argv[1] through argv[5], in that order.

Again, times are converted to minutes. If the ON time is earlier than the OFF time, the ON time falls within the next day, so 24 hours (24 * 60 minutes) are added to the ON time. If ON and OFF times are the same, AfterHours allows the request. If the time now falls between the OFF and ON times, AfterHours selects argv[1] for output in the message template. If the time now falls outside that period, AfterHours simply returns STATUS_PPP_OK to indicate that the event may proceed.

If the event is to be disallowed, this is achieved by combining the message template with the selected message in a call to fprintf, sending output to STDOUT. The action returns STATUS_PPP_OK to allow the event to proceed. User Manager applies the stylesheet ppp.xsl, which is part of the COREid System, and the resulting page containing the text message is returned to the browser.

A MANAGEDLIB Action Example

The following is a sample header file that declares class EventAPI.

```
// managed_ppp.h

#ifndef __managed_ppp__
#define __managed_ppp__

#using <mscorlib.dll>
#using <pppinterface.dll>
#using <System.dll>
#using <System.Xml.dll>

using namespace System;
using namespace System::Text;
using namespace System::Collections;
using namespace System::Xml;
using namespace System::Net;
using namespace System::IO;
using namespace Oblix::Identity::CoreID;
```

```
/* Singleton class that contains methods specified as Identity Event API actions.
   Netpoint will instantiate one EventAPI object, which will be shared among
   threads.
   Class members must be accessed in a thread-safe manner. Modification of data
   members must be synchronized.

   The class must be named EventAPI and must define a constructor and destructor
*/

public __gc class EventAPI {

public:
   /* ctor, initialize class members here */
   EventAPI( );
   /* dtor, release resources here */
   virtual ~EventAPI( );

   /* action methods */
   IPPPData::STATUS_PPP_M PreProcessingTest( String * eventName, IPPPData * data
);
   IPPPData::STATUS_PPP_M PostProcessingTest( String * eventName , IPPPData *
data );
   IPPPData::STATUS_PPP_M PostProcessingTest_Phone( String * eventName, IPPPData
* data );
   IPPPData::STATUS_PPP_M SavePreProcessing( String * eventName, IPPPData * data
);
   IPPPData::STATUS_PPP_M WorkflowPreActionTest( String * eventName, IPPPData *
data );
   IPPPData::STATUS_PPP_M WorkflowPostActionTest( String * eventName , IPPPData *
data );
   IPPPData::STATUS_PPP_M WorkflowPostActionPasswordTest( String * eventName,
IPPPData * data );
   IPPPData::STATUS_PPP_M WorkflowExtActionTest( String * eventName, IPPPData *
data );
   IPPPData::STATUS_PPP_M WorkflowSubflowActionTest( String * eventName,
IPPPData * data );
   IPPPData::STATUS_PPP_M PasswordTest( String * eventName, IPPPData * data );
   IPPPData::STATUS_PPP_M WorkflowRetryTest( String * eventName, IPPPData * data
);
   IPPPData::STATUS_PPP_M NISClient( String * eventName, IPPPData * data );
   IPPPData::STATUS_PPP_M ProcessCPResponseEncryption( String * eventName,
IPPPData * data );
   IPPPData::STATUS_PPP_M USCOnChange( String * eventName, IPPPData * data );
   IPPPData::STATUS_PPP_M NavigationTest( String * eventName, IPPPData * data );

private:
   String * MakePayload( IPPPData * data, String * login, String * password,
String * group , String * user );
};


public __gc class XMLUtil {
```

```
public:
   static String * XMLUtil::knewline = S"\n";
   static String * XMLUtil::kSpace= S" ";
   static String * XMLUtil::kCloseAngle = S">";
   static String * XMLUtil::kProcessingInst = S"<?xml version=\"1.0\"?>";
   static String * XMLUtil::kSoapEnvEnvelopeStart = S"<SOAP-ENV:Envelope";
   static String * XMLUtil::kSoapEnvEnvelopeEnd = S"</SOAP-ENV:Envelope>";
   static String * XMLUtil::kxmlns = S"xmlns:oblix=\"http://www.oblix.com\"
xmlns:SOAP-ENV=\"http://schemas-xmlsoap.org/soap/envelope/\"";
   static String * XMLUtil::kSoapEnvBodyStart = S"<SOAP-ENV:Body>";
   static String * XMLUtil::kSoapEnvBodyEnd = S"</SOAP-ENV:Body>";
   static String * XMLUtil::kOblixAuthStart = S"<oblix:authentication ";
   static String * XMLUtil::kOblixAuthEnd = S"</oblix:authentication>";
   static String * XMLUtil::kTypeBasic = S"type=\"basic\"";
   static String * XMLUtil::kObLoginStart = S"<oblix:login>";
   static String * XMLUtil::kObLoginEnd = S"</oblix:login>";
   static String * XMLUtil::kObPasswordStart = S"<oblix:password>";
   static String * XMLUtil::kObPasswordEnd= S"</oblix:password>";
   static String * XMLUtil::kObReqStart = S"<oblix:request";
   static String * XMLUtil::kObReqEnd = S"</oblix:request>";
   static String * XMLUtil::kApp = S"application=\"groupservcenter\"";
   static String * XMLUtil::kfuncname = S"function=\"subscribeUserToGroup\"";
   static String * XMLUtil::kObParamsStart= S"<oblix:params>";
   static String * XMLUtil::kObParamsEnd = S"</oblix:params>";
   static String * XMLUtil::kObParamStart = S"<oblix:param";
   static String * XMLUtil::kObParamEnd = S"</oblix:param>";
   static String * XMLUtil::kNameEqproxysourceuid = S"name=\"proxysourceuid\"";
   static String * XMLUtil::kNameEquid = S"name=\"uid\"";

};

#endif
```

There are several “action” methods in the class. The directive

```
#using <pppinterface.dll>
```

indicates that the plug-in will be using date types from pppInterface.dll, namely the IPPPDate interface as well as the status codes. Initialization code should be placed in the constructor, EventAPI(), and clean-up code should be placed in the destructor ~EventAPI().

The sample plug-in also uses data types from the System.Xml library, as indicated by the directive

```
#using<System.Xml.dll>
```

The method EventAPI::NISClient is an example of how to send a SOAP request using classes from System.Xml, which is part of the .NET framework SDK. The method subscribes the target user to a group. The target user is obtained through IPPPDate::Get, while other parameters, including the group to which to subscribe the user, are obtained from a configuration file, params.xml.

The method uses these parameters to construct the SOAP request with the method EventAPI::MakePayLoad (not listed here, but it is in the sample code). It then creates an http request with the URI parameter using HttpWebRequest. It then gets a stream from that request and writes the SOAP request (IdentityXML/ subscribeUserToGroup) to the stream. Afterwards, the method gets a response from the request.

```
IPPPData::STATUS_PPP_M
EventAPI::NISClient( String * eventName, IPPPData * data )
{
   IPPPData::STATUS_PPP_M retStatus = IPPPData::STATUS_PPP_M::STATUS_PPP_OK;

   try {
      String * sUidParamName = S"proxysourceuid";
      String * sGroupDNParamName = S"uid";
      String * uri, * login, * password, * group, * user;
      String * errMsg = S"Missing Parameter";

      String * targets[] = data->Get( S"WfInstance.obtargetdn" );
      user = targets[0];
      XmlDocument& doc = *new XmlDocument;
      doc.Load( S"params.xml" );
      XmlNode * root = doc.FirstChild;
      XmlElement * elem;
      elem = root->get_Item( S"uri" );
      if( elem != NULL ) { uri = elem->InnerText; } else { throw new Exception(
errMsg ); }
      elem = root->get_Item( S"login" );
      if( elem != NULL ) { login = elem->InnerText; } else { throw new Exception(
errMsg ); }
      elem = root->get_Item( S"password" );
      if( elem != NULL ) { password = elem->InnerText; } else { throw new
Exception( errMsg ); }
      elem = root->get_Item( S"group" );
      if( elem != NULL ) { group = elem->InnerText; } else { throw new Exception(
errMsg ); }

      String * sPayLoad = MakePayload( data, login , password , group , user);
      XmlDocument& soapReq = *new XmlDocument;
      soapReq.LoadXml( sPayLoad );

      HttpWebRequest * req = static_cast<HttpWebRequest*>( WebRequest::Create(
uri ) );
      req->ContentType = "text/xml;charset=\"utf-8\"";
      req->Accept = "text/xml";
      req->Method = "POST";

      Stream * stm = req->GetRequestStream( );
      soapReq.Save( stm );
      stm->Close( );

      WebResponse * resp = req->GetResponse( );
```

```
   }
   catch( Exception * e ) {
      data->SetResultString( e->ToString( ) );
      retStatus = IPPPData::STATUS_PPP_M::STATUS_PPP_ABORT;
   }

   return retStatus;
}
```

Parameter File:

```
params.xml
<Root>
  <uri>http://sdelaney/identity/oblix/apps/groupservcenter/bin/
groupservcenter.cgi</uri>
  <login>admin</login>
  <password>oblix</password>
  <group>cn=Group of Employees10k1 with 1000 members, ou=Corporate,
o=Company,c=US</group>
</Root>
```

5 Building AccessGates with the Access Server SDK

This chapter describes the Access Server SDK and how you use it to create custom AccessGates. It is organized around the following sections:

- "About AccessGates" on page 246 introduces AccessGates and their role in the NetPoint system. The section also discusses AccessGate architecture.
- "About AccessGate Deployment" on page 251 explains the tasks you must complete to create and enable an AccessGate.
- "About the Access Server SDK" on page 265 describes the directory structure and content of the installed SDK.
- "About the Access Server API" on page 268 helps you select an AccessGate development platform by comparing the development language-specific implementations of each class in the Access Server API.
- "About Custom AccessGate Code" on page 284 explains how to write each functional section of code that goes into a typical AccessGate.
- "C++ Implementation Details" on page 337 presents reference details for the C++ implementation of the Access Server API.
- "C Implementation Details" on page 352 presents reference details for the C implementation of the Access Server API.
- "C# Implementation Details" on page 368 presents reference details for the C# implementation of the Access Server API.
- "Java Implementation Details" on page 382 presents reference details for the Java implementation of the Access Server API.
- "Best Practices" on page 399 presents suggestions on how to avoid problems with your AccessGate. It also presents tips for identifying and resolving the most common AccessGate problems.

About AccessGates

AccessGates are Access Server clients or agents. They process user requests for access to resources within the LDAP domain protected by your NetPoint system.

Typically, you embed custom AccessGate code in a servlet (plug-in) or stand-alone application that receives resource requests. This code uses Access Server API libraries to perform authentication and authorization services on the Access Server.

If a resource is not protected, the AccessGate grants the user free access to the requested resource. If the resource is protected and the user is authorized to provide certain credentials to gain access, the AccessGate attempts to retrieve those user credentials so that the Access Server can validate them. If authentication of the user and authorization for the resource succeed, the AccessGate makes the resource available to the user.

Note: For the purposes of this document, "Access Server API" refers narrowly to the set of programming calls that enable developers to access the authentication, authorization, and other services of a NetPoint Access Server. By contrast, "Access Server SDK" refers to all the files installed by the Access Server SDK installation package.

About Prefabricated AccessGates (WebGates)

NetPoint ships with several prefabricated AccessGates known as WebGates. Each of these out-of-the box WebGates has been set up to protect HTTP resources on a specific web server such as:

- Microsoft Internet Information Server
- iPlanet/SunONE Web Server
- Apache Web Server
- Lotus Domino
- IBM HTTP Server (IHS)

Some WebGates can protect Embedded Java Bean (EJB) resources (which are non-HTTP resources) on application servers such as BEA WebLogic and IBM WebSphere. For a matrix listing the WebGate implementations available for various combinations of host server software and host machine operating system, see the *NetPoint 7.0 Installation Guide*.

When to Create a Custom AccessGate

Typically, you deploy a custom AccessGate instead of a standard WebGate when you need to control access to a resource for which NetPoint does not already supply an out-of-the-box solution. This might include:

- Protection for non-HTTP resources
- Protection for a custom web server developed to implement a special feature (such as reverse proxy, for example)
- Implementation of single sign-on (SSO) to protect a combination of HTTP and non-HTTP resources

  Example: you can create an AccessGate that facilitates SSO within an enterprise environment that includes a WebLogic cluster as well as non-WebLogic resources.

AccessGate Architecture

Each AccessGate is built from three types of resources:

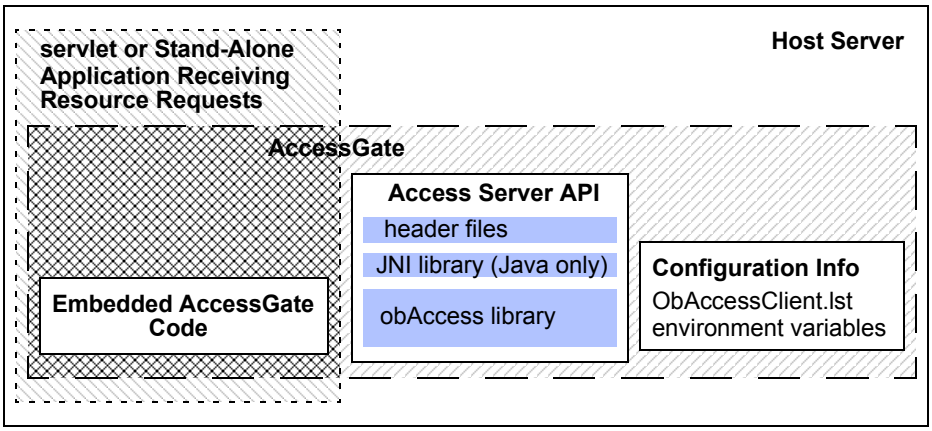
- Custom AccessGate code, which you build into a servlet or stand-alone application running on the machine where the rest of the AccessGate resides. You can write AccessGate code using any of four development language platforms:
  - C++
  - C (pseudo object-oriented classes)
  - C# (.NET framework managed code)
  - Java

  These platforms, which support equivalent functionality implemented in language-specific ways, serve as interfaces to the underlying NetPoint code, which is written in C++.
- Configuration information, which consists of the following:
  - Environment variables, which you set on the server where the AccessGate is installed. These variables differ, depending on whether your server runs UNIX or Windows.
  - An ObAccessClient.lst file, which is stored on the server where the AccessGate is installed. This file contains configuration information entered through the “configureAccessGate” command-line application.
  - AccessGate connection settings that you enter, view, and edit by navigating to Access System Console > AccessGate Configuration. These settings are stored in your Oblix configuration directory.

- The various implementations of the Access Server API libraries, which facilitate AccessGate interaction with the Access Server. These include:
  - Header files for either Java or the “C-family” languages (C\C++\C#)
  - The JNI library (for Java only, packaged in jobaccess.jar)
  - The ObAccess library (specific to the operating system platform used by the machine hosting the AccessGate)

The following diagram shows AccessGate components installed on a host server:

Figure 4 Architectural Detail of an AccessGate



AccessGate Variations

AccessGates can differ according to a variety of factors:

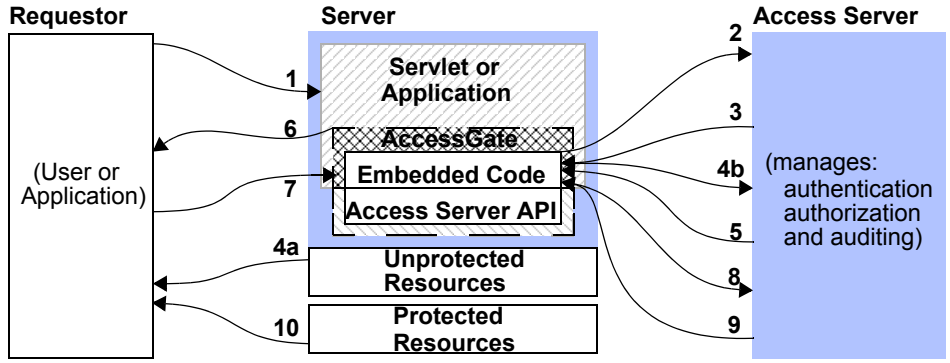
- The operating system of the host machine on which they are installed (Each OS platform requires a different Access Server SDK installation package.)
- Whether they run as stand-alone applications or server plug-ins
- The development language in which they are written (These development languages provide a choice of interfaces to the underlying functionality of the API)
- The type of server for which they are written (You can protect web servers or application servers)
- The type of resources they protect (You can protect both HTTP and non-HTTP resources)
- They ways in which they retrieve user credentials (You can enable HTTP FORM-based input, the use of NetPoint session tokens, and command-line input, among other methods)

How an AccessGate Handles a Resource Request

Regardless of the variability introduced by the preceding factors, most AccessGates follow the same basic steps to process user requests.

When a user or application submits a resource request to a servlet or application running on the server where the AccessGate is installed, the AccessGate code embedded in that servlet or application initiates the basic process shown in the following diagram:

Figure 5 Process Overview: Handling a Resource Request



Process Overview: Handling a resource request

1. The application or servlet containing the AccessGate code receives a user request for a resource.
2. The AccessGate constructs a ObResourceRequest structure, which the AccessGate code uses when it asks the Access Server whether the requested resource is protected.
3. The Access Server responds.
4. If the resource is not protected,
   a) The AccessGate grants the user access to the resource. Otherwise...
   b) The AccessGate constructs an ObAuthenticationScheme structure, which it uses to ask the Access Server what credentials the user needs to supply. (This step is only necessary if the AccessGate supports the use of different authentication schemes for different resources).
5. The Access Server responds.
6. The application uses a form or some other means to ask the user for her credentials. In some cases, the user credentials may already have been submitted as part of:
   - A valid session token
   - Input from a web browser

- Arguments to the command-line script or keyboard input that launched the AccessGate application

7. The user responds to the application.
8. The AccessGate constructs an ObUserSession structure, which presents the user credentials to the Access Server, which maps them to a user profile in the NetPoint user directory.
9. If the credentials prove valid, the AccessGate creates a session token for the user, then it sends a request for authorization to the Access Server. This request contains the user identity, the name of the target resource, and the requested operation.
10. The AccessGate grants the user access to the resource, providing, of course, that the user is authorized for the requested operation on the particular resource.
11. (Not pictured). A well-behaved AccessGate deallocates the memory used by the objects it has created, then shuts down the Access Server API.

The preceding steps represent only the main path of the authorization process. Typically, additional code sections within the servlet or application handle branch situations where:

- The requested resource is not protected
- The authentication challenge method associated with the protected resource is not supported by the application
- The user has a valid Oblix single sign-on cookie (ObSSOCookie), which allows the user to access to the resource without re-presenting her credentials for as long as the NetPoint session token embedded in the cookie remains valid. For details about ObSSOCookies and single sign-on, see the *NetPoint 7.0 Administration Guide Volume 2*.
- The user fails to supply valid credentials under the specified conditions
- Some other error condition arises
- The developer has built additional custom code into the AccessGate to handle special situations or functionality

About AccessGate Deployment

AccessGates are typically deployed by teams, with each team member covering a specific area of expertise. For instance, a network administrator can install software and set the requisite environment variables, a developer can write the custom AccessGate code, and a Netpoint Access Administrator can create policy domains to protect specific resources. Together, the developer and the Access Administrator can configure the Access Server to work with the new AccessGate.

Although the tasks handled by each individual can vary, the team responsible for a NetPoint system must complete the following tasks.

Task overview: AccessGate deployment

1. Install the Access Server SDK on the machine that will host the AccessGate, as described in"Installing the Access Server SDK" on page 252.
2. Write custom AccessGate code and build it into a servlet or application that receives resource requests, as described in "Writing AccessGate Code" on page 264.
3. Configure the AccessGate, as described in "Configuring an AccessGate" on page 259 and includes the following:
   - Set environment variables on the host server where the AccessGate will be installed.
   - Create an AccessGate entry on the Access System console (typically, the Access Administrator and the AccessGate developer work together to create this entry).
   - Create an ObAccessClient.lst file, which the AccessGate developer accomplishes by running the interactive, non-GUI *configureAccessGate* application on the machine that will host the AccessGate.
4. Protect enterprise resources by creating policy domains, as described in the *NetPoint 7.0 Administration Guide Volume 2*.

   This includes definition of the resource and the designation of operations permitted against that resource. Generally, a NetPoint Access Administrator performs this task through the Access Manager console.

Note: The Access Administrator and the developer must work closely to ensure that the resource types and challenge methods the AccessGate is programmed to handle match exactly the resource types and challenge methods assigned to the policy domains that the AccessGate will protect. For details about protecting resources with policy domains, see the *NetPoint 7.0 Administration Guide Volume 2*.

Installing the Access Server SDK

If your NetPoint system uses WebGates exclusively, you do not need to install the Access Server SDK, because each self-contained WebGate installation package already contains all the specific resources it needs.

If your NetPoint system has one or more custom AccessGates, you must install an instance of the Access Server SDK on each server that hosts an AccessGate. You may install both Unix-based and Windows-based AccessGates within the same NetPoint system as long as each instance of the Access Server SDK matches the type of server on which it is installed.

Note that the Access Server SDK is not part of the Access Server installation package. The self-contained SDK ships in its own setup package which is labelled:

```
NetPoint#_#_Platform_AccessServerSDK[.ext]
```

where #_# is the installed version of NetPoint, *platform* is the operating system of the host server where you install the SDK, and *ext* is the file name extension ".exe," which appears only on Windows installation packages.

Thus, "NetPoint6_5_sparc-s2_AccessServerSDK" contains the version of the Access Server SDK appropriate for installation on servers running the Solaris operating system.

A convenient installation location for your Access Server SDK on a typical Windows system might be:

```
C:\Program Files\Oblix\AccessServerSDK
```

In any case, make note of the SDK installation path, as you will need it later, whenever you see the string *SDK_install_dir* in the rest of this chapter.

For more information, see:

- "Access Server SDK Compatibility" on page 252
- "Access Server API and Access Manager API Support" on page 254
- "Obtaining the Access Server SDK" on page 254

Access Server SDK Compatibility

For the most up-to-date platform support, see the *NetPoint 7.0 Release Notes* and the Oblix Customer Care Web site, which includes details about our support policy:

https://customers.oblix.com/shared/prodinfo/prod_roadmap.cfm

Table 2 shows OS support for the Access Server SDK.

Table 2 OS Support for Access Server SDK

Operating System	Access Server SDK
Red Hat Linux Enterprise AS 3.0	Access API Access Manager API
Red Hat Linux Enterprise AS 2.1	SHAREid support
Sun Solaris 8	Access API Access Manager API
Microsoft Windows 2000 Advanced Server SP4	Access API Access Manager API
Microsoft Windows Server 2003 (Enterprise Edition)	Access API Access Manager API

The NetPoint Access Server SDK operates with the following JDK versions:

- JDK v1.2.2
- JDK v1.3.1
- JDK v1.4

Red Hat Linux AS 3.0 Prerequisites

The ASDK 7.0 sample C++ programs need GCC 3.3.2 compiler. No compat libraries need to be installed when using ASDK 7.0 on Linux AS 3. Instead the GCC compiler needs to be upgraded to GCC 3.3.2

- **Runtime Requirements**—For Java ASDK, Sun JVM 1.4.1 or IBM JVM 1.4.1 are recommended.
- **JVM and JDK Requirements**—Sun v1.4.1 and IBM v1.4.1 are supported.

To compile sample programs

1. Use the supported GCC compiler v3.3.2.
2. Locate the sample file in *install_dir*/samples/makefile.sample.
3. Change JAVA_HOME=<IBM/SUN JDK 1.4.1>.

Access Server API and Access Manager API Support

Table 3 shows supported platforms/compilers for Access Server API and Access Manager API. Any commercially available C or Java (JDK 1.2.2, 1.3.1, or 1.4) compiler is supported. The COM/COM+ interface is no longer supported; functionality has been replaced with a ManagedCode API.

Table 3 OS Support for Access Server API and Access Manager API

Operating System	API Platform/Compiler and COM/COM+ interface
Solaris Linux	GCC 3.3.2
Sun Solaris 8 Solaris 9	GNU 3.2.2 (API platform/compiler)
Microsoft Windows 2000 Advanced Server SP4	Visual Studio 6.x (API platform/compiler)
Microsoft Windows Server 2003 (Enterprise Edition)	Visual Studio .NET 7.0 Visual Studio .NET 7.0 (.NET Managed Code)
Red Hat Enterprise Linux AS v2.1	GNU 3.2.2 (API platform/compiler)

For the latest support information, see:

https://customers.oblix.com/shared/prodinfo/prod_roadmap.cfm

Obtaining the Access Server SDK

You can obtain a copy of the Access Server SDK installation package from these sources:

- The Oblix Web site
- The Oblix FTP server

To obtain the installation package from the Oblix Web site

1. Using an account with administrator privileges, log onto the machine where you plan to install your AccessGate, then launch your web browser and navigate to the following web site:

   ```
   http://www.oblix.com
   ```

2. At the top of the page, click the tab marked Customer Services.
3. When the next screen appears, click the down arrow to the right of the list box marked “select resource.”
4. Click “customers.oblix.com.”

5. In the field marked Username, type in the user name assigned to your company when you purchased NetPoint. In the field marked Password, type the password associated with that user name. Click “login” to advance to the next screen.
6. Click the Downloads tab at the top of the page.
7. When the next screen appears, click the version of NetPoint for which you are building a custom AccessGate.
8. When the next screen appears, locate the section of the page marked “Software,” then click the operating system of the machine where the AccessGate will reside.
9. When the next screen appears, locate the section of the page marked “The Access System,” then click the link labelled Access Server SDK.
10. When the pop-up window appears, follow the directions to save the installation package to a hard disk on the machine where the AccessGate will reside.

To obtain the installation package from the Oblix FTP server

1. Follow the first 5 steps in the preceding procedure.
2. Click the “My Company” tab at the top of the page.
3. Find the table titled “General Company” and write down the values listed for “FTP UserName” and “FTP Password.”
4. Click the “Downloads” tab at the top of the page.
5. When the next screen appears, click “Product Patches.”
6. When the next screen appears, click the link marked as follows:

   NetPointVersion Patch Information

   where *NetPointVersion* is the version of NetPoint you are using.
7. When the next page appears, click the link marked “Click here for patches.”
8. When a new window opens with “ftp://ftp.oblix.com” in the address bar, click the directory whose name most closely matches the version of NetPoint you are using.

   You may have to click one or more additional layers of folders to reach the release number your want. (To reach NetPoint 7.0, for example, you click “NetPoint6.x,” then “NetPoint 7.0” In this case, you also need to click “FCS1.1a,” which represents general release version for NetPoint 7.0).
9. When the next screen appears, click the folder representing the operating system platform of the server on which your AccessGate will reside.
10. When the next screen appears, click the folder named “AccessSystem.”

11. Copy the following file to the hard disk of the machine where your AccessGate will reside:

    *NetPoint_Platform*_AccessServerSDK[.ext]

    where *NetPoint* is the installed version of NetPoint, *platform* is the operating system of the host server where you install the SDK, and *ext* is the Windows-only file name extension for the installation package.

Installing the SDK on Windows

To Install the SDK on a Windows Machine

1. On the machine where you plan to install your AccessGate, navigate to the directory where your Access Server SDK installation package is stored. The path to this package resembles the following:

   Device\...\AccessSystem*Platform*

   where *Device* is the CD or hard drive holding your NetPoint installation image, and *Platform* is the operating system of the host server on which you are installing the AccessGate.

2. Double-click the following executable:

   NetPoint*#_#_Platform*_AccessServerSDK

   where #_# is the version of NetPoint you are running, and *Platform* is the operating system of the machine on which you are installing the AccessGate.

   For Example:

   NetPoint7_0_Win32_AccessServerSDK.exe

3. When the Welcome screen appears, click next.
4. When the license agreement appears, decide whether to proceed by checking the box "I accept the terms of the license agreement."
5. The next screen emphasizes that you must have administrator privileges on the host machine where you are installing the SDK.

   If your current account has administrator rights, click Next.

   If you are not currently logged onto an account with such privileges, complete the following sub-task:

   a) Click Cancel to close the installation wizard

   b) Log off the system

   c) Log back on using an administrator account

   d) Restart the Access Server SDK installation wizard

6. Select an install directory using any of the following methods:

- Click Browse and navigate to the directory you prefer
- Place your cursor in the “Destination Name” entry field and type the path to the directory you prefer
- Simply accept the default installation directory as it appears in the “Destination Name” entry field (For Windows, the default is C:\Program Files\NetPoint. The sub-directory \AccessServerSDK is appended to the default base path during installation).

In any case, when the directory you want appears in the “Destination Name” entry field, click Next to continue.

7. When a screen appears to announce the target installation directory, check to make sure that it shows the exact location you want. Make a note of this path, because you will need it every time *SDK_install_dir* appears in the rest of this chapter.
8. Click next to commence file installation.
9. Respond to the on-screen prompts, as necessary.
10. When installation completes, a screen appears to report that the process has succeeded.

Installing the SDK on Unix

To Install the SDK on a Unix Machine

1. On the machine where you plan to install your AccessGate, navigate to the directory where your Access Server SDK installation package is stored.

   The path to this package resembles the following:

   ```
   Device\...\AccessSystem\Platform
   ```

   where *Device* is the CD or hard drive holding your NetPoint installation image, and *Platform* is the operating system of the machine on which you are installing the AccessGate.

2. Locate the following executable:

   ```
   NetPoint#_#_Platform_AccessServerSDK
   ```

   where #_# is the version of NetPoint you are running, and *Platform* is the operating system of the machine on which you are installing the AccessGate.

   For example:

   ```
   NetPoint7_0_sparc-s2_AccessServerSDK
   ```

3. At the UNIX prompt, enter the name of the appropriate Access Server SDK installation package to commence GUI-mode installation.

   Note: This procedure assumes that your UNIX machine supports GUI-mode. You can also run the installation package in interactive command-line mode by entering the following:

   ```
   run ./installationPackage
   ```

   where *installationPackage* is the name of the Access Server SAK installation Package appropriate for your machine.

4. When the Welcome screen appears, click next.
5. When the license agreement appears, decide whether to proceed by checking the box "I accept the terms of the license agreement."
6. When the installer asks for a user and group to set as the owner of the installed files, you may find it convenient to specify the same user and group that "own" the server application your AccessGate will protect. In any case, you must be logged on as the user you specify, or as "root," in order to continue installation.
7. Accept the default install directory by hitting Return, or type your preference, then hit return.

   Note: You cannot install NetPoint components in any directory that contains special characters in its path. The proscribed characters are: blank spaces, new lines, *, [], {}, and so on.

8. When a screen appears to announce the target installation directory, check to make sure that it shows the exact location you want. If the directory does not exist, the installer creates it. Make a note of this path, because you will need it every time *SDK_install_dir* appears in the rest of this chapter.
9. Click next to commence file installation.
10. Respond to the on-screen prompts, as necessary.
11. When installation completes, a screen appears to report that the process has succeeded.

Configuring an AccessGate

AccessGate configuration, which is not to be confused with AccessGate creation or SDK installation, consists of the following sub-tasks.

Task overview: Configuring an AccessGate

1. Setting environment variables on the host server where the AccessGate will reside, as described in “Setting Environment Variables” on page 259.
2. Creating an AccessGate entry on the Access Server, as described in “Creating an AccessGate Entry on the Access Server” on page 261.
3. Creating the ObAccessClient.lst file within the Access Server SDK installation, as described in “Running the configureAccessGate Utility” on page 262.
4. For each AccessGate on your NetPoint system, you must run the *configureAccessGate* utility, as described in “Running the configureAccessGate Utility” on page 262.

Note: You can perform any of the AccessGate configuration either before or after you create custom code for your AccessGate.

Setting Environment Variables

Requisite environment variables differ according to the operating system on the host server where your AccessGate resides. Complete the procedure below that is appropriate to your environment:

- Windows host
- Unix host

To Set Environment Variables on a Windows Machine

1. Navigate to Start Menu > Control Panel > System > Advanced > Environment Variables.
2. Examine the contents of the System Variables box.
3. If you see the Variable Name for the Variable Value you want to add, click that Variable Name, click Edit, then proceed to Step 6 (otherwise, click New and proceed to the next step).
4. Type Variable Name and Variable Value in the appropriate fields of the New System Variable entry box.
5. Click OK to commit the variable, then proceed to Step 7.
6. When the Edit System Variable entry box appears, click the Variable Value field, move the cursor to the end of the string, type “; *value*” (semi-colon

followed by the blank space character followed by the new value), then click OK to commit the variable.

7. Repeat Steps 3-6 until you have added all the variables listed in the following table.

Important: For Windows 2003, these variables take effect immediately and do not require a system reboot.For Windows 2000, you should reboot your machine after you have entered the variables. This ensures that all the variables will take effect.

Table 4 Windows Environment Variables

Variable Name = *existing path*; Value to Add	Description
PATH = *existing path*; *SDK_install_dir*\oblix\lib	Points to obaccess.dll and other library files.
CLASSPATH = *existing path*; *SDK_install_dir*\oblix\lib\jobaccess.jar	Points to the name and location of the Java class archive for the Access Server API. (Required only if you use the Java implementation of the Access Server API to write custom AccessGate code).
OBACCESS_INSTALL_DIR = SDK_install_dir	Points to the Access Server SDK install root. (This is necessary only if your AccessGate does not specify *SDK_install_dir* as part of the ObConfig.initialize method).

To Set Environment Variables on a Unix Machine

1. Use a text editor to open the file (or files) containing the variables on your UNIX system.
2. For all the variables in the following table, append the values listed, or, if the variable name does not exist, add the variable name along with its associated value to the file.

Important: To ensure that the new variables take effect, take whatever measures (such as system reboot) are appropriate for your specific UNIX environment.

Table 5 UNIX Environment Variables

Variable Name = *existing path*; Value to Add	Description
LD_LIBRARY_PATH = *existing path*; *SDK_install_dir*/oblix/lib (for Solaris only)	Points to libobaccess.so and other library files on Solaris systems.

Table 5 UNIX Environment Variables

Variable Name = *existing path*; Value to Add	Description
CLASSPATH = *existing path*; *SDK_install_dir*/oblix/lib/jobaccess.jar	Points to the name and location of the Java class archive for the Access Server API. (Required only if you use the Java implementation of the Access Server API to write custom AccessGate code).
POST_CLASSPATH = *existing path*; *SDK_install_dir*/oblix/lib/jobaccess.jar	
OBACCESS_INSTALL_DIR = *existing path*; *SDK_install_dir*	Points to the Access Server SDK install root.

Creating an AccessGate Entry on the Access Server

Complete the following task to enable your Access Server to connect to your custom AccessGate. (You can complete this task before your create your AccessGate, as long as the information you enter matches the particulars of the AccessGate and the ObAccessClient.lst file).

To Create an AccessGate Entry on the Access Server

1. Navigate to NetPoint System Console > Access System Configuration > Add New AccessGate.
2. Type a convenient name in the AccessGate Name field.

   Note: Choose a name that distinguishes this particular AccessGate from all the others in your NetPoint system. For instance, "CustCare5_6006" might help you identify an AccessGate installed on web server "Customer Care 5," which listens on port 6006.

3. In the HostName field, type the DNS name of the machine hosting the server instance on which the AccessGate resides.

   For Example:

   CustomerCare5.oblix.com

4. Complete the activitiy below based on your environment:

   - If the machine on which your will install the AccessGate does *not* host additional Web or applications servers skip this step.

     Typically, this value is assigned by the administrator responsible for the server; the AccessGate administrator merely records this value in the AccessGate confuguration profile).

   - If the machine on which your will install the AccessGate hosts does host additional Web or applications servers, specify the server instance that will

use your AccessGate by typing in the port number the server uses to listen for user requests.

Note: Oblix recommends using any number between 6000 and 65,536, which has not been used for any other ports on the network.

5. Type an alphanumeric string for use as a password whenever the AccessGate connects to the Access Server.

   This value is optional for all transport modes, although the "simple" and "cert" modes use other passwords not directly related to AccessGate configuration. However, Oblix strongly recommends that you set a password for your AccessGate, particularly if it uses Open mode. This will prevent unauthorized AccessGates from connecting to Access Servers.

6. Retype the password to confirm it.

7. Click Save at the bottom of the panel to commit the values.

The preceding steps provide all necessary information for this stage of AccessGate deployment. From this point forward, you can enter optional values for the other parameters, which will use the supplied default values until you replace them. For details on setting these other parameters by modifying an AccessGate, see the *NetPoint 7.0 Administration Guide Volume 2*.

Running the configureAccessGate Utility

For each AccessGate on your NetPoint system, you must run the "configureAccessGate" utility, which stores data used to initialize the AccessGate in the file ObAccessClient.lst.

The configureAccessGate tool reads and updates this information each time you initialize the Access Server API and at other points during AccessGate operation.

By default, this file is stored in the following directory on the machine hosting your AccessGate:

SDK_install_dir\oblix\config

You can view the contents of the ObAccessClient.lst file by opening it in any text editor. For details on the content of the ObAccessClient.lst file and modifying an AccessGate, see the *NetPoint 7.0 Administration Guide Volume 2*.

Important: Never edit ObAccessClient.lst using a text editor. Instead, use the *configureAccessGate* application from a command-line window.

Using the *configureAccessGate* application from a command-line window to edit ObAccessClient.lst ensures that your AccessGate parameters remain consistent throughout the NetPoint system. This is because confgureAccessGate not only modifies ObAccessClient.lst, it performs additional vital tasks related to the simple and cert modes, such as creating or requesting an X.509 certificate. See details on modifying an AccessGate in the *NetPoint 7.0 Administration Guide Volume 2*.

To run configureAccessGate on a UNIX Machine

1. From the UNIX command line, navigate to the following directory:

   SDK_install_dir\oblix\tools

   where *SDK_install_dir* is the root directory of your Access Server SDK installation.

2. Type the following command, then press Enter:

   ```
   ./configureAccessGate -i SDK_install_dir -t AccessGate
   ```

3. Respond to the series of prompts as they appear.

   For information on available switches, acceptable arguments, and defaults when configuring AccessGates, see the *NetPoint 7.0 Administration Guide Volume 2*.

When the configureAccessGate program successfully exits, the AccessGate is enabled on your server.

To run configureAccessGate.exe on a Windows Machine

1. Navigate to Start Menu > Run.
2. Enter the following command in the Open field:

   cmd

3. When the command-line (non-GUI) window opens, switch directories by entering the following command:

   cd *SDK_install_dir*\oblix\tools\configureAccessGate

   where *SDK_install_dir* is the path to your Access Server SDK installation.

4. Launch the “configureAccessGate” utility by entering the following command, including switches and arguments:

   ```
   configureAccessGate -i SDK_install_dir -t AccessGate
   ```

   where *SDK_install_dir* is the path to your Access Server SDK installation.

Respond to the series of prompts as they appear. For information on available switches, acceptable arguments, and defaults when configuring AccessGates, see the *NetPoint 7.0 Administration Guide Volume 2*.

When the configureAccessGate program successfully exits, the AccessGate is enabled on your server.

Writing AccessGate Code

This procedure is covered in the section “About Custom AccessGate Code” on page 284.

Cloning a Custom AccessGate

When you need to protect an additional server by creating an AccessGate similar to an AccessGate you have already deployed, you do not necessarily have to write new code. In some cases, you might be able to clone the existing AccessGate to that additional server.

Task Overview: Cloning a custom AccessGate

1. Install the version of the Access Server SDK that is compatible with the operating system of the server where the cloned AccessGate will reside. See “Access Server SDK Compatibility” on page 252.
2. Create an entry for the new AccessGate on the Access Server to which it will connect. See “Creating an AccessGate Entry on the Access Server” on page 261.
3. Run the configureAccessGate utility to create an ObAccessClient.lst file for the new AccessGate on the server where the new AccessGate will reside. See “Running the configureAccessGate Utility” on page 262.

   Alternatively, you modify the ObAccessClient.lst file from the original AccessGate by copying it to the new host server, then running the “configureAccessGate” utility.
4. Set the Access Server API environment variables for the server on which the cloned AccessGate will reside. See “Setting Environment Variables” on page 259.
5. Copy the plain text file containing your custom AccessGate code to the server where the cloned AccessGate will reside.
6. Modify the transferred code, as necessary, to fit the particulars of the new AccessGate.
7. Recompile the code using the compiler that is compatible with the operating system of the host server and the development language in which the custom AccessGate code was written. See “Access Server SDK Compatibility” on page 252.

Protecting Resources

Policy domains specify which resources are protected by what protection methods applied to which users and groups. Usually, they are created and maintained by NetPoint Access Administrators. AccessGate developers should work with Access administrators to create, modify, or identify the specific policy domains that the AccessGate will protect. For detailed information on creating policy domains, see the *NetPoint 7.0 Administration Guide Volume 2*.

About the Access Server SDK

This section begins with an overview of the Access Server SDK. Next, it outlines the content of the installed SDK directory and subdirectories. Finally, it introduces the set of files supplied to support custom AccessGates for BEA WebLogic systems.

SDK Overview

The Access Server SDK is an optional component of the NetPoint system, and is installed independent of the Access Server. The SDK provides all the information and resources you need to build a custom AccessGate. In addition to the files that make up the various implementations of the Access Server API, the SDK includes documentation and code samples, which show how to construct simple AccessGate servlets or applications for each of the supported development platforms.

SDK Content

The Access Server SDK installation directory contains the following subdirectories and content:

_jvmAccessSDK—Contains the Java runtime resources used by the Access Server SDK install wizard.

_uninstAccessSDK—Contains the resources for uninstalling the Access Server SDK through the install wizard.

apidoc—Documents the Java implementation of the Access Server API. You access this information through the following URL:

`SDK_install_dir\apidoc\com\oblix\access\package-summary.html`

Note: The package “com.oblix.access” documents the Access Server API. The package “com.oblix.accessmgr” documents the Access Management API. See “Access Management API” on page 403.

---

examples—Includes a sample build file, a sample "make" script, and the web page "AppServer_ReadMe.html," which explains how to create an AccessGate for a WebLogic server.

obaccess—Contains an example Java servlet as well as prototypes for classes that can be used to extend the startup and shutdown classes for the server application.

ejbAccessTest— Holds the example "Broker Bean" EJB, along with a sample "build" file and a "build" script which follows the conventions used by BEA WebLogic.

include—Contains header files that define the classes, methods, and functions composing the C++ and C implementations of the Access Server API.

oblix—Holds four subdirectories:

config—Contains configuration data for the Access Server SDK installation.

lang—Contains language-specific files (English, French, etc.) for your installation of the Access Server SDK, including:

- **release notes**—Contains information that was not received in time to include in the main documentation. For late-breaking information, be sure to check the Oblix Customer Care website at the following URL:

  `http://www.oblix.com/services/index.html`

- **netlibmsg.lst**—A file of messages the AccessGate uses when errors occur. Depending on how your AccessGate is configured, these can be logged locally, displayed locally, or ignored.
- **ObAccessClient.msg**—Which provides the message text the AccessGate displays in response to various events.

lib—Contains the Access Server SDK libraries and Java archive files that are built into the application. These include:

- **various libraries**—Codes libraries required by the API. (For instance, ".dll" files for Windows, ".so" files for Solaris, etc.)
- **jobaccess.jar**—Which is the Java archive file for the API.
- **ObAccessClient.lst file**—An example of the AccessGate configuration file.

orig—Contains information created during the installation of the SDK. You should ignore this.

tools—Contains four significant subdirectories:

- **configureAccessGate**—Contains the tool that configures the AccessGate, as well as the messages the tool needs.

- **lang_tools**—Contains development language-specific files (Java, C, etc.) for your installation of the Access Server SDK.
- **migration_tools**—Contains information about migrating AccessGates created with earlier versions of the API to the current version of the SDK.
- **openssl**—Contains a tool and short certificate used to configure the AccessGate for "simple" or "cert" mode operation.

samples—Provides versions of the standalone Access Test application written for Java, C, C++, and C# (.NET). These can be used to gain familiarity with the SDK build process before you attempt more complex applications.

Important: Once the SDK is installed, do not change the relative locations of the subdirectories and files. Doing so may prevent an accurate build and proper operation of the API.

BEA WebLogic Support Files

To enable quick support of BEA WebLogic, the Access Server SDK provides WebLogic-compatible Startup and Shutdown classes. The WebLogic Server API itself provides extensible classes that can be used to extend the startup and shutdown capabilities of the WebLogic Server.

The ObStartupAppGate.java and ObShutdownAppGate.java classes reside in:

```
SDK_install_dir/examples/obaccess
```

These classes enable WebLogic Server initialization, which is necessary before AccessGate initialization can take place. They can be used out-of-the-box or modified to suit the application being developed.

The ObStartupAppGate and ObShutdownAppGate Java classes conform to the standards detailed in the WebLogic documentation supplied by BEA. Before writing your own Startup and Shutdown classes for the WebLogic Server, refer to the section on server startup and shutdown in the "NetPoint Access Server SDK and WebLogic Application Server" documentation, which is available in *SDK_install_dir*\examples\ AppServer_Readme.html.

The Access Server API method "ObConfig.initialize" initializes the AccessGate. The "ObConfig.shutdown" method shuts down the AccessGate gracefully. After successful initialization, all Java components built using the Access Server SDK share the AccessGate configuration when they are deployed to the WebLogic Server.

About the Access Server API

This section begins by comparing the naming schemes used by each of the four Access Server API implementations. It then introduces the API classes in turn, with special emphasis on the different ways specific features are handled in the various development environments.

Reference details for each of the four language-specific implementations appear in separate sections beginning on page 337. For the Java implementation only, reference details are also supplied as online JavaDoc HTML files accessible through *SDK_install_dir*/apidoc/index-all.html.

Implementations Compared

The Access Server API allows developers to write custom AccessGate code in any of four development languages: Java, C, C++, or C#. While each of these implementations takes advantage of platform-specific features to implement the the API, the four implementations are functionally equivalent.

About Memory Management

The four implementations of the Access Server API differ most significantly in the area of memory management.

Java and C# both feature automatic "garbage collection." Neither language allows you to call a destructor explicitly. Instead, you simply let the built-in garbage collector deallocate the memory for unused objects when it (the garbage collector) deems appropriate. Thus, the garbage collectors do not guarantee when an object will be cleaned up, but they do see to it that all objects are destroyed when they are no longer referenced, and no memory leak occurs.

By contrast, you must explicitly call destructors in C and C++ to clean up the objects that your program no longer needs. For each C-language "pseudo class," you use the functions whose names end with "_free." When you no longer need C++ objects previously created with the "new" operator, use the "delete" operator to destroy them.

Corresponding Classes

The functionality of the Access Server API has been organized into seven basic classes. Even for the C language, which is not explicitly object-oriented, the functions have been organized into "pseudo object-oriented classes."

The following table lists the corresponding class names for each language platform:

Table 6 Access Server API Implementation-specific Classes Compared

Purpose of the Class	C++	C	C#	Java
Supports parameter storage structures (lists or hashtables).	ObMap	ObMap_t	ObDictionary	java.util.Hashtable, which extends java.util.Dictionary (This is not a Com.Oblix.Access class)
Supports iteration within lists (C and C++ implementations only; C# and Java enumerate hashtables).	ObMapIterator	ObMap Iterator_t	ObDictionary Enumerator	
Creates and manipulates structures that handle user authentication.	ObAuthentication Scheme	ObAuthn Scheme_t	ObAuthentication SchemeMgd	ObAuthentication Scheme implements ObAuthentication Scheme Interface
Creates and manipulates structures that handle user requests for resources.	ObResource Request	ObResource Request_t	ObResource RequestMgd	ObResource Request implements ObResource RequestInterface
Creates and manipulates structures that handle user sessions, which begin when the user authenticates and end when the user logs off or the session times out.	ObUserSession	ObUser Session_t	ObUser SessionMgd	ObUserSession implements ObUserSession Interface
Retrieves and modifies AccessGate configuration information.	ObConfig	ObConfig_t	ObConfigMgd	ObConfig
Handles errors thrown by the Access Server API	ObAccess Exception	ObAccess Exception_t	ObAccess ExceptionMgd	ObAccess Exception

About Multi-Language Implementation

You can select any of the four functionally-equivalent implementations of the Access Server API as the development language interface you use to write your custom AccessGate code. However, you should remain aware that your code, no matter what language it was written in, will interact with underlying C++ binaries in the API.

Also, AccessGate code created for one development language/compiler/server OS platform combination might require recompilation to ensure that it will run correctly on another combination.

To ensure that your AccessGates behave as expected, you should follow certain “best practices” in the following areas:

- **Portability**—See the recommendations in “Cloning a Custom AccessGate” on page 264.
- **Clean-up**—See the section “About Memory Management” on page 268.

ObMap

When your AccessGate interacts with the Access Server, it stores, passes, and receives information through lists of entries (or “items”) arranged as name:value pairs. These list structures, which are opaque to the end user, are also known as “maps” in the C and C++ environments. When used within a Java or C# context, they are called “hashtables.”

These list and hashtable structures store many types of Access Server API-related data, including the following:

- Resource request information
- Authentication scheme information
- User session information
- AccessGate configuration information

For instance, a typical AccessGate might pass a set of user credentials to the Access Server as a single-item list in the following form:

```
UserName=JSmith&Password=J5m1th
```

The C and C++ implementations of the Access Server API create and manipulate these structures through the ObMap class and ObMap_t “pseudo class,” respectively. The equivalent class for the C# implementation is ObDictionary. The Java implementation of the API does not include a class of its own to handle list structures; rather, it relies on the standard Java class named “java.util.Hashtable” for all list-related functions.

All of these implementing classes provide methods to enable the following functionality:

- Create a list (or hashtable)
- Add a name:value pair to a list (or hashtable)
- Read a name:value pair from the list (or hashtable) when the name half of the item is known
- Report the total number of items in a list (or hashtable)
- Copy an existing list (or hashtable)
- Deallocate the memory used by the list structure (or hashtable)

For a discussion of additional methods that manipulate Access Server API lists and hashtables, see “ObMapIterator” on page 271

Equivalent Methods

The following table presents equivalent constructors and methods for the four API implementations of the ObMap class. Note that this table includes existing Java methods only if they correspond to an equivalent method in one of the “C-family” implementations of ObAccess:

Table 7 Methods (and Constructors) for the ObMap Class Compared

C++ (ObMap)	C (ObMap_t)	C# (ObDictionary)	Java (java.util.Hashtable)
get	ObMap_get	get_Item	get
put	ObMap_put	add	put
size	ObMap_size	get_Count	size
copy	ObMap_copy	Clone	Hashtable(map t)
Delete	ObMap_free	(built-in garbage collection)	(built-in garbage collection)
(constructor)	ObMap_new	(constructor)	(constructor)

ObMapIterator

Sometimes, it is necessary to step through (or iterate) the items in a hastable or list. The C, C++, and C# implementations handle this and related functions through the ObMapIterator, ObMapIterator_t, and ObDictionary classes, respectively. The Java implementation achieves this functionality through java.util.Hashtable, which is a standard Java class, rather than an Access Server API class.

The methods offered by the implementations differ because only the C and C++ implementations of the Access Server API require full iterator functionality to parse their list structures. The C# and Java implementations of the API use hashtables, and therefore do not use the same iterator functionality for parsing operations.

ObMapIterator provides methods to enable the following pointer functionality:

- Create a list pointer
- Move the pointer from the current item to the next item in the list
- Determine whether additional items exist in the list beyond the position currently occupied by the pointer. By implication, you know that the pointer

has reached the end of a list when no more items exist beyond the current position of the pointer.

- Deallocate the memory used by the pointer.

Equivalent Methods

The following table presents equivalent constructors and methods for the four API implementations of the ObMapIterator class. Note that this table includes Java methods only if they correspond to an equivalent method in one of the "C-family" implementations of ObAccess.

Table 8 Methods for the ObMapIterator Class Compared

C++ (ObMapIterator)	C (ObMapIterator_t)	C# (ObDictionary Enumerator)	Java (java.util.Hashtable)
next	ObMapIterator_next	MoveNext	
		get_Current	
hasMore	ObMapIterator_hasMore		
		get_Entry	
		get_Key	
		get_Value	
(constructor)	ObMapIterator_new	Reset	
Delete	ObMapIterator_free	(built-in garbage collection)	(built-in garbage collection)

ObAuthenticationScheme

The Access Server API creates ObAuthenticationScheme structures to store, pass, and retrieve information about the authentication scheme (authentication template) associated with the target resource requested by a particular user. In other words, an authentication scheme specifies how a user is to be challenged for a set of credentials.

The details for each authentication scheme are specified when the Access Administrator creates a policy domain on the Access Server to protect a specific resource. For a detailed discussion of authentication schemes, see the *NetPoint 7.0 Administration Guide Volume 2*.

Credentials are name:value pairs that the AccessGate passes to the Access Server in order to authenticate a user. For example, an AccessGate using the HTTP basic challenge method might pass the following credential string, which contains two name:value pairs:

```
userid=JSmith&Password=J5m1th
```

In the preceding example, the name:value pairs are separated by the ampersand character (&), and the name and value components are separated by equal signs (=).

Since the requisite authentication scheme can vary according to the resource requested, an ObAuthenticationScheme structure can be created only after an ObResourceRequest structure has specified the target resource.

Each authentication scheme contains the elements listed in the following table:

Table 9 ObAuthenticationScheme Elements

<table>
<tr><th>Element</th><th colspan="3">Details</th></tr>
<tr><td>Display Name</td><td colspan="3">This is a friendly name used to identify the authentication scheme. For example, “Customer Form Login” might represent an authentication scheme used to grant preferred customers access to a price list for frequent buyers.</td></tr>
<tr><td>Mask Byte</td><td colspan="3">This byte indicates the type of challenge method to be used and whether credentials need to be sent over a secure connection:</td></tr>
<tr><td></td><td>Challenge Method</td><td>Mask</td><td>Expected Credentials</td></tr>
<tr><td></td><td>none</td><td>0x00</td><td>No credentials needed. The plug-in should map to an anonymous user.</td></tr>
<tr><td></td><td>basic</td><td>0x01</td><td>Userid and Password (as for HTTP basic)</td></tr>
<tr><td></td><td>certificate</td><td>0x02</td><td>A certificate via SSL/TLS client authentication (as for HTTPS)</td></tr>
<tr><td></td><td>form</td><td>0x04</td><td>Customer-defined credential fields in an HTML login form</td></tr>
<tr><td></td><td>secure</td><td>0x08</td><td>Credentials must be sent over a secure connection (as for HTTPS) and a redirection URL must be used as well.</td></tr>
<tr><td>Strength</td><td colspan="3">This positive integer defines the level of authentication.</td></tr>
<tr><td>Redirection URL</td><td colspan="3">This is the URL (in the form “https://host:port”) where HTTP secure authentication is to be performed. If secure authentication (or a central authentication server such as SecurID) is not required, this value is set to NULL.</td></tr>
</table>

Table 9 ObAuthenticationScheme Elements

Element	Details		
Challenge Parameters	This element stores additional authentication scheme-related information in name:value pairs. When these optional parameters are not supplied, Challenge Parameters is represented by an empty string.		
	Parameter Name	**Challenge Method**	**Value**
	realm	basic	The authentication domain (as for an LDAP directory)
	form	form	The URL of the login form that will be displayed on the user web browser
	creds	form	A space-separated list of login form fields that will be used as credentials
	action	form	The URL to which the login form posts the data it receives
Plug-in Sequence	This element is not visible through the Access Server API.		

Equivalent Methods

The following table presents equivalent constructors and methods for the four API implementations of the ObAuthenticationScheme class.

Table 10 Methods for the ObAuthenticationScheme Class Compared

C++ (ObAuthentication Scheme)	C (ObAuthentication Scheme_t)	C# (ObAuthentication SchemeMgd)	Java (ObAuthentication Scheme)
getName	ObAuthn_getName	get_Name	getName
getMask	ObAuthn_getMask	get_Mask	(this method is not public in Java)
requires SecureTransport	ObAuthn_requires SecureTransport	get_Requires SecureTransport	requires SecureTransport
IsBasic	ObAuthn_isBasic	get_IsBasic	isBasic
IsCertificate	ObAuthn_isCertificate	get_IsCertificate	isCertificate
IsForm	ObAuthn_isForm	get_IsForm	isForm
IsNone	ObAuthn_isNone	get_IsNone	isNone
getLevel	ObAuthn_getLevel	get_Level	getLevel

Table 10 Methods for the ObAuthenticationScheme Class Compared

C++ (ObAuthentication Scheme)	C (ObAuthentication Scheme_t)	C# (ObAuthentication SchemeMgd)	Java (ObAuthentication Scheme)
getRedirectUrl	ObAuthn_get RedirectUrl	get_RedirectUrl	getRedirectUrl
getChallenge Parameter	ObAuthn_get ChallengeParameter	get_Challenge Parameter	getChallenge Parameter
getAllChallenge Parameters	ObAuthn_getAll ChallengeParameters	get_All ChallengeParameters	getAllChallenge Parameters
getNumberOf Challenge Parameters	ObAuthn_get NumberOf ChallengeParameters	get_NumberOf ChallengeParameters	getNumberOf Challenge Parameters
(constructor)	ObAuthn_new	(constructor)	(constructor)
(copy constructor)	(not implemented)	Clone	clone
Delete	ObAuthn_free	(built-in garbage collection)	(built-in garbage collection)

ObResourceRequest

The Access Server API uses the ObResourceRequest structure to store, pass, and retrieve information concerning a user request for access to a resource. This information includes the elements listed in the following table:

Table 11 ObResourceRequest Elements

Element	Details
Resource Type	This can be a built-in type, such as HTTP or EJB, or a custom type defined through the Access System Console. For a detailed discussion about configuring resource types and protecting resources with policy domains, see the *NetPoint 7.0 Administration Guide Volume 2*.
Resource Name	The name of the target resource within the NetPoint name space. This must be provided in the format `[//host[:port]]/resourceName` where the optional *host* and *port* values indicate the Web server servicing *resourceName*, which is the name of the targeted resource. Host and port apply only to HTTP resources.

Table 11 ObResourceRequest Elements

Element	Details
Operation	The action to be performed against the resource, as dictated by the resource type. Examples are GET and POST for HTTP resources, and EXECUTE for EJB resources. For custom resource types, operations are defined through the Access System Console when the resource type is defined. For a detailed discussion, about configuring resource types and protecting resources with policy domains, see the *NetPoint 7.0 Administration Guide Volume 2*.
Parameter Set (optional)	A name:value pair for the requested operation. Parameter names and values must be strings. For HTTP resources, they can be extracted from the request query string or POST data. For EJB resources, parameter entries can be “bean” method parameters. Neither of the preceding is a requirement. The names and values can be any arbitrary data that the developer and Access Administrator have agreed upon. The name:value pairs can be used to supply data for authorization requests. This is useful for authorizations that require data from external sources. For example, if you need to pass an account number, you can write a plug-in for this purpose. For details about customizing access control with plug-ins, see the *NetPoint 7.0 Customization Guide*.

The ObResourceRequest constructors return the following policy information from the Access Server:

Table 12 Information Returned by the Access Server in Response to ObResourceRequest

Element	Details
Protection Flag	Indicates whether the resource request is protected by NetPoint policies. If the resource is not protected, the AccessGate grants the user free access to the resource.
Authentication scheme name	An internal ID representing the authentication scheme associated with the target resource.

ObAuthenticationScheme constructors use the information contained in the ObResourceRequest structure to determine which authentication scheme is associated with the target resource. Similarly, the ObUserSession constructors can use the information in the ObResourceRequest structure to determine whether the user, once authenticated, is authorized to access the target resource.

Equivalent Methods

The following table lists the names of equivalent methods across the four implementations of the ObResourceRequest class.

Table 13 Methods for the ObResourceRequest Class Compared

C++ (ObResource Request)	C (ObResource Request_t)	C# (ObResource RequestMgd)	Java (ObResource Request)
getResourceType	ObResource_ getResourceType	get_ResourceType	getResourceType
getResource	ObResource_getResource	get_Resource	getResource
getOperation	ObResource_getOperation	get_Operation	getOperation
getParameters	ObResource_ getParameters	get_Parameters	getParameters
getNumberOf Parameters	ObResource_getNumber OfParameters	get_NumberOf Parameters	getNumberOf Parameters
isProtected	ObResource_isProtected	get_IsProtected	isProtected
getAuthorization Parameters	ObResource_ getParameters	get_Authorization Parameters	getAuthorization Parameters
getNumberOf Authorization Parameters	ObResource_getNumber OfAuthorizationParameters	get_NumberOf Authorization Parameters	getNumberOf Authorization Parameters
(copy constructor)	(not implemented)	Clone	Clone
Delete	ObMap_free	(built-in garbage collection)	(built-in garbage collection)
(constructor)	ObResourceRequest_new	(constructor)	(constructor)

ObUserSession

After the Access Server validates the user credentials necessary for successful login (or “authentication”), the AccessGate creates an ObUserSession structure to store, pass, and retrieve information about the user, the target resource, and various types of authentication policy information. This structure can be created from information drawn from the ObResourceRequest and ObAuthenticationScheme structures as well as information returned by the Access Server. Alternatively, the ObUserSession structure can be created from information contained in a session token, which is an ASCII character string that stores information about a currently valid user session.

A session token can be generated from a ObUserSession structure. An ObUserSession structure can be constructed from a valid session token, *except* for actions and error information, which are not carried in the session token. For details about context-specific data for an authentication request and form-based authentication to pass the originally requested URL to a change password servlet, see the *NetPoint 7.0 Administration Guide Volume 2*.

A key method in ObUserSession returns information from the Access Server as to whether the user, who has passed authentication successfully, is authorized to access the target resource. Other ObUserSession methods return information about when the user authenticated (and, by extension, when the current session will expire), the most recent time at which the user was authorized to access a resource, and so forth. In any case, a ObUserSession structure contains the pieces of information presented in the following table:

Table 14 ObUserSession Elements

Element	Description
User Identity	The Distinguished Name (DN) of the user's profile entry in the LDAP user directory.
Level	The security level of the authentication scheme used to authenticate the user. This is a relative number assigned by the Access System Administrator. For additional discussion of changing the security level of an authentication scheme, see the *NetPoint 7.0 Administration Guide Volume 2*.
Location (optional)	The location of the user web browser (or the proxy server representing the user's web browser). Examples are the DNS hostname of the proxy server or the IP address of the user's browser.
Session start time	The time when the user authenticated. This and the maximum permitted session time can be used to calculate when the session expires.
Last use time	The most recent time at which the user was authorized; this is used to determine when an "idle session" expires.
Actions	Actions set during authentication and authorization according to NetPoint policy rules. Each rule includes a type, which can be user created, that tells the application how the action is to be interpreted. Examples for HTTP are "cookie" and "headerVar."
Status	The current status of the session, which may be one of the following: • logged in • logged out • login failed • expired
Error number and Error Message	Errors resulting from the most recent authentication or authorization.

Equivalent Methods

The following table presents the names of equivalent methods for the ObUserSession class in the Access Server API.

Table 15 Methods for the ObUserSession Class Compared

C++ (ObUserSession)	C (ObUserSession_t)	C# (ObUserSessionMgd)	Java (ObUserSession)
getLocation	ObUser_getLocation	get_Location	getLocation
getAction	ObUser_getAction	getAction	getAction
getActions	ObUser_getActions	getActions	getActions
getActionTypes	ObUser_ getActionTypes	get_ActionTypes	getActionTypes
getNumberOfActions	ObUser_ getNumberOfActions	getNumberOfActions	getNumberOfActions
getLevel	ObUser_getLevel	get_Level	getLevel
getStartTime	ObUser_ getStartTime	get_StartTime	getStartTime
getLastUseTime	ObUser_ getLastUseTime	get_LastUseTime	getLastUseTime
getStatus	ObUser_getStatus	get_Status	getStatus
getUserIdentity	ObUser_ getUserIdentity	get_UserIdentity	getUserIdentity
getError	ObUser_getError	get_Error	getError
getErrorMessage	ObUser_ getErrorMessage	get_ErrorMessage	getErrorMessage
isAuthorized	ObUser_isAuthorized	isAuthorized	IsAuthorized
isAuthorizedWith Parameters	ObUser_isAuthorized WithParameters	isAuthorized WithParameters	IsAuthorized (with additional parameters)
getSessionToken	ObUser_ getSessionToken	get_SessionToken	getSessionToken
setLocation	ObUser_setLocation	set_Location	setLocation
(copy constructor)	(not implemented)	Clone	Clone
logoff	ObUser_logoff	LogOff	logoff

Table 15 Methods for the ObUserSession Class Compared

C++ (ObUserSession)	C (ObUserSession_t)	C# (ObUserSessionMgd)	Java (ObUserSession)
Delete	ObUser_free	(built-in garbage collection)	(built-in garbage collection)
ObUserSession (constructor)	ObUserSession_fromToken ObUserSession_Authenticate	ObUserSessionMgd (constructor)	ObUserSession (constructor)

ObConfig

The ObConfig class includes methods to initialize and shut down the Access Server API as well as store, pass, retrieve, and in some cases, modify, configuration data for the AccessGate.

The ObConfig.initialize method does the following:

- Passes the name of the installation directory to the AccessGate after retrieving that value from either the “installDir” parameter or the environment variable OBACCESS_INSTALL_DIR
- Verifies that the ObAccessClient.lst file exists in the Access Server installation directory and is readable by the AccessGate
- Reads the bootstrap (current) AccessGate configuration from ObAccessClient.lst
- Opens the ObAccessGate.msg message catalog to obtain the text to be used for user errors and exceptions
- Connects to one or more Access Servers as specified in the bootstrap configuration
- Obtains the full AccessGate configuration from the Access Server
- Creates the local resource request and authentication scheme caches
- Creates a thread to update the AccessGate configuration periodically

ObConfig also contains a shutdown method that you must call to release resources when an application no longer needs to use the Access Server API.

Configuration Parameters

The following table details the configuration information maintained for each AccessGate. These items are read into the ObConfig structure each time the AccessGate is initialized. They can be Accessed through ObConfig.getItem and ObConfig.getAllItems in the C++ implementation. For the C, C#, and Java implementations, the corresponding methods are ObConfig_getItem and ObConfig_getAllItems, ObConfigMgd.getItem and ObConfigMgd.getAllItems, and Com.Oblix.Access.getItem and Com.Oblix.Access.getAllItems, respectively.

Table 16 AccessGate Configuration Parameters

Parameter Name	Parameter Value
accessServerTimeout	The number of seconds that a connection to an Access Server is left open before the connection is re-established.
cacheTimeout	The number of seconds that a cached authentication scheme or resource request object can exist before being flushed automatically. A value of zero specifies that cached elements should never be flushed.
debug	On or Off. If debug is on, the AccessGate traces all messages sent to Access Servers.
failoverThreshold	If the number of primary Access Servers connected to the AccessGate falls below this threshold, the Access Server API opens one or more connections to secondary Access Servers.
id	The string identifier for the AccessGate in the Oblix configuration directory.
idleTimeout	The maximum number of seconds allowed to elapse between authorizations. When this value is exceeded, the user needs to authenticate again.
transportSecurity	One of the following security modes used to connect to the Access Servers: **open** — no encryption **simple** — TLS encryption, using certificates generated from a built-in CA **cert** — TLS encryption, using certificates issued by a full CA
lastUpdateTime	The number of seconds between 1/1/1970 00:00 and the most recent time the AccessGate configuration was updated.
maxCacheElements	The maximum number of resource-request objects in the authentication scheme cache, which is of a fixed size.
maxConnections	The maximum number of connections that can be opened to AccessGates.

Table 16 AccessGate Configuration Parameters

Parameter Name	Parameter Value
preferredHost	The Web server host address to which the user's browser is redirected when an authentication scheme requires secure authentication. For example, the AccessGate uses this value to specify the host in the authorization request.
primaryDomain	The domain used to set ObSSOCookies, as for a single sign-on domain. Other applications are free to interpret or ignore this parameter, as needed.

primary_server_list	A list of Access Servers to which the AccessGate connects first.
	The list follows the form:
	`host1:port1,numConn1, host2:port2,numConn2`
	`...`
	where *hostn* is the DNS of the Access Server.

secondary_server _list	List of Access Servers to which the AccessGate connects if the number of connections to the primary servers falls below the failoverThreshold.
	The list follows the form:
	`host1:port1,numConn1, host2:port2,numConn2`
	`...`
	where *hostn* is the DNS of the Access Server.

sessionTimeout	The maximum number of seconds a user session created by the application remains valid.
sleepFor	How often (in seconds) the AccessGate checks to confirm that the Access Server connections are up.
state	**enabled or disabled**. Interpretation of this parameter is up to the application. When disabled, an AccessGate immediately allows access to all resources.

Equivalent Methods

The following table presents the names of equivalent methods for the ObConfig class in the Access Server API.

Table 17 Methods for the ObConfig Class Compared

C++ ObConfig	C ObConfig_t	C# ObConfigMgd	Java ObConfig
initialize	ObConfig_initialize	initialize	initialize
shutdown	ObConfig_shutdown	shutdown	shutdown
getAllItems	ObConfig_getAllItems	get_AllItems	getAllItems

Table 17 Methods for the ObConfig Class Compared

C++ ObConfig	C ObConfig_t	C# ObConfigMgd	Java ObConfig
getNumberOfItems	ObConfig_ getNumberOfItems	get_NumberOfItems	getNumberOfItems
getItem	ObConfig_getItem	getItem	getItem
getSDKVersion	ObConfig_getSDKVersion	get_SDKVersion	getSDKVersion
getNAPVersion	ObConfig_getNAPVersion	get_NAPVersion	getNAPVersion

ObAccessException

When the Access Server API methods detect problems, they throw an ObAccessException. The kind of error that has occurred is deducible from the enumerated list of C-family error message names beginning on page 397. (The Java equivalents are on page 390).

Depending upon the particular error, zero to five substrings of data can be inserted into the exception message text provided in the ObAccessGate.msg catalog. (This insertion feature applies only to the C-family of implementations; the Java message strings must be handled as indivisible units).

For example, ObAccessException_ NOT_PROTECTED, error code 208, is defined as follows:

```
ObAccessException_NOT_PROTECTED {
   Unprotected resource %1 used in an
   ObAuthenticationScheme or ObUserSession
   constructor.}
```

If this error occurs while the application is processing an unprotected resource named “xresource,” the API builds an ObAccessException, whose structure contains the error code 208 and the text “xresource,” which replaces the %1 substring.

C-family methods for this class allow you to extract the error code and the substring, by its index (1 to 5). You can also generate a string equal to the entire message with the substring(s) inserted. In the C++ environment, you should delete the ObAccessException that it catches.

By comparison, the Java implementation is limited, because it only supports the retrieval of entire messages. In other words, you cannot extract or otherwise manipulate substrings.

The C implementation of ObAccessException requires you to write an exception handler to trap errors. See “C-language Error Handlers” on page 366 for a full discussion. Such an error handler is implemented in the sample program “Example: access_test_c.cpp” on page 291.

Equivalent Methods

The following table lists equivalent methods across the four implementations of the ObAccessException class.

Table 18 Methods for the ObResourceRequest Class Compared

C++ (ObAccess Exception)	C (ObAccess Exception)	C# (ObAccess ExceptionMgd)	Java (ObResource Request)
getCode	ObAccessException_getCode	get_Code	
getParameter	ObAccessException_getParameter	getParameter	
toString	ObAccessException_toString	get_String	
getCodeString	ObAccessException_getCodeString (now deprecated; see page 366)	getCodeString	
(constructor)	Exception Handler (registered callback function)	(constructor)	(constructor)

About Custom AccessGate Code

The structure of a typical AccessGate application roughly mirrors the sequence of events required to set up an AccessGate session.

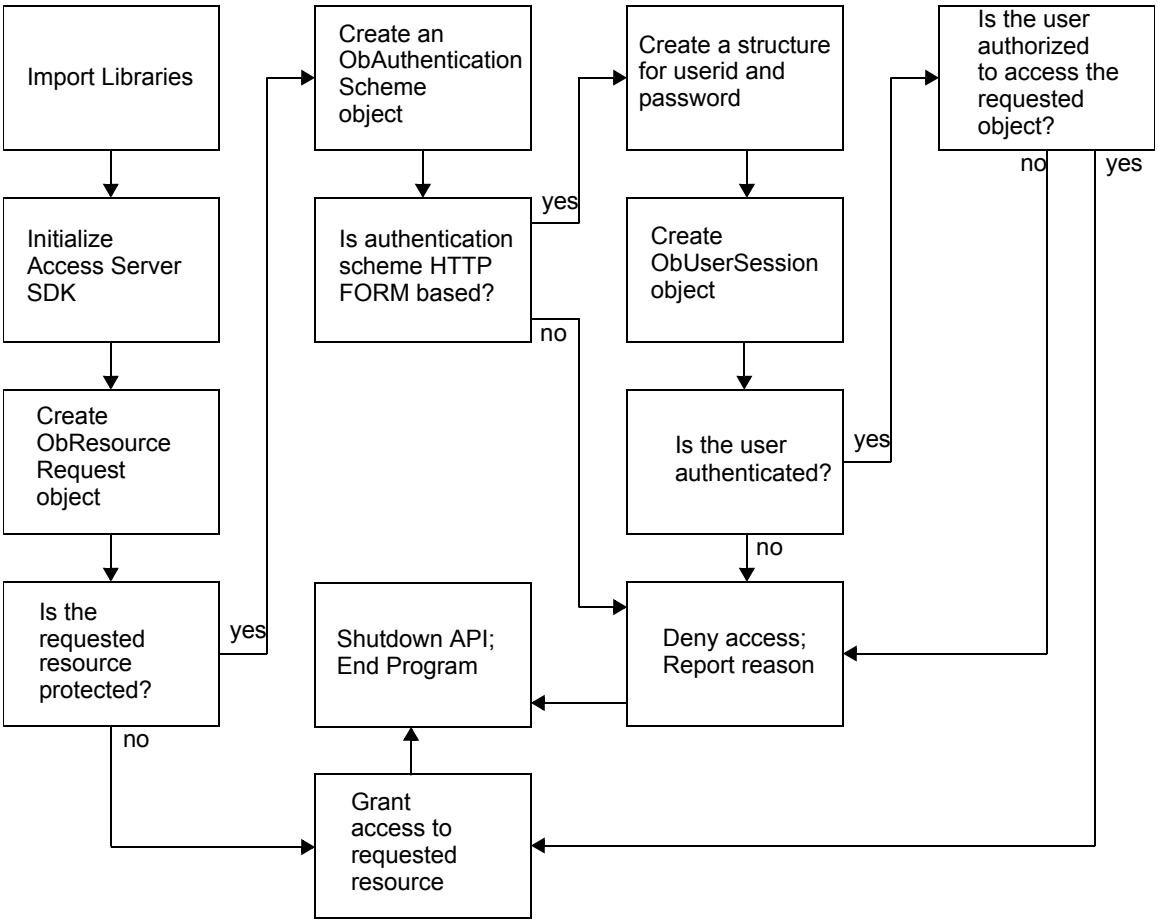
AccessGate application structure contains the following sections

1. Include or import requisite libraries
2. Get resource
3. Get authentication scheme
4. Gather user credentials required by authentication scheme
5. Create user session
6. Check user authorization for resource
7. Clean up (for C and C++ only: C# and Java use automatic garbage collection)
8. Shut down

Typical AccessGate Execution Flow

All HTTP FORM-based AccessGate applications and plug-ins follow the same basic pattern, as illustrated by the following figure:

Figure 6 A Minimal AccessGate built for FORM-based authentication.



Note: To run this test application, or any of the other examples, in this chapter, you must make sure that your Access System is installed and set up correctly. Specifically, check that it has been configured to protect resources that match exactly the URLs and authentication schemes expected by the sample programs. For details on creating policy domains and protecting resources with policy domains, see the *NetPoint 7.0 Administration Guide Volume 2*.

Example: JAccessGate.java

This example is the simplest AccessGate program presented in this document. It illustrates how to implement the bare minimum tasks required for a working AccessGate:

- Connect to the Access Server
- Log in using an authentication scheme employing the HTTP FORM challenge method
- Check authorization for a certain resource using an HTTP GET request
- Catch and report Access Server API exceptions

Typically, this calling sequence is quite similar among AccessGates using the FORM challenge method. FORM-method AccessGates differ principally in the credentials they require for authentication and the type of resources they protect.

A complete listing for JAccessGate.java appears immediately below. You can copy this code verbatim into the text file JAccessGate.java and execute it on the machine where your Access Server SDK is installed. The subsection that follows the listing annotates the code line-by-line so that developers can become familiar with the Java versions of the Access Server API calls.

Listing 29 JAccessGate.java

```
import java.io.*;
import java.util.*;
import java.text.*;
import com.oblix.access.*;

public class JAccessGate {
   public static final String ms_resource = "//Example.com:80/secrets/
        index.html";
   public static final String ms_protocol = "http";
   public static final String ms_method = "GET";
   public static final String ms_login = "jsmith";
   public static final String ms_passwd = "j5m1th";
   public static void main(String argv[]) {
      try {
         ObConfig.initialize();
         ObResourceRequest rrq = new ObResourceRequest(ms_protocol, ms_resource,
              ms_method);
         if (rrq.isProtected()) {
            System.out.println("Resource is protected.");
            ObAuthenticationScheme authnScheme = new ObAuthenticationScheme(rrq);
            if (authnScheme.isForm()) {
               System.out.println("Form Authentication Scheme.");
               Hashtable creds = new Hashtable();
               creds.put("userid", ms_login);
               creds.put("password", ms_passwd);
```

```
            ObUserSession session = new ObUserSession(rrq, creds);
            if (session.getStatus() == ObUserSession.LOGGEDIN) {
               if (session.isAuthorized(rrq)) {
                  System.out.println("User is logged in and authorized for the
                        request at level " + session.getLevel());
               } else {
                  System.out.println("User is logged in but NOT authorized");
               }
            } else {
               System.out.println("User is NOT logged in");
            }
         } else {
            System.out.println("non-Form Authentication Scheme.");
         }
      } else {
         System.out.println("Resource is NOT protected.");
      }
   }
   catch (ObAccessException oe) {
      System.out.println("NetPoint Access Exception: " + oe.getMessage());
   }
   ObConfig.shutdown();
  }
}
```

Annotated Code

Import three standard Java libraries to provide system input and output, text handling, and other basic functions.

```
import java.io.*;
import java.util.*;
import java.text.*;
```

Import the library containing the Java implementation of the Access Server API classes. To ensure that these libraries are visible, check that the CLASSPATH environment variable (for both UNIX and Windows platforms) points to the directory containing jobaccess.jar, which is installed by default in *SDK_install_dir*/oblix/lib.

```
import com.oblix.access.*;
```

This application is named *JAccessGate*.

```
public class JAccessGate {
```

Since this is the simplest of example applications, we are declaring global constants to represent the parameters associated with a user request for access to a resource.

Typically, a real-world application receives this set of parameters as an array of strings passed from a requesting application, HTTP FORM-based input, or command-line input.

```
public static final String ms_resource = "//Example.com:80/secrets/
     index.html";
public static final String ms_protocol = "http";
public static final String ms_method = "GET";
public static final String ms_login = "jsmith";
public static final String ms_passwd = "j5m1th";
```

Launch the main method on the Java interpreter. An array of strings, which is named "argv," gets passed to the main method. In this particular case, the user "jsmith," whose password is "j5m1th," has requested the HTTP resource //Example.com:80/secrets/index.html. GET is the specific HTTP operation that will be performed against the requested resource. For details about supported HTTP operations and protecting resources with policy domains chapter of the *NetPoint 7.0 Administration Guide Volume 2*.

```
public static void main(String argv[]) {
```

Place all relevant program statements in the main method within a large *try* block so that any exceptions will be caught by the *catch* block at the end of the program.

```
   try {
```

Initialize the Access Server SDK so that both the "Com.Oblix.Access" Java classes and the native JNI objects are available to the JAccessGate application. Since you do not specify the SDK installation root here, we use the value stored in the OBACCESS_INSTALL_DIR environment variable.

You only need to initialize the SDK once, but the initialization must occur before you attempt any calls to the Access Server API.

```
      ObConfig.initialize();
```

Create a new resource request object named "rrq" using the ObResourceRequest constructor with the following three parameters:

- **ms_protocol**, which represents the type of resource being requested. When left unspecified, the default value is HTTP. EJB is another possible value, although this particular example does not cover such a case. You can also create custom types, as described in the *NetPoint 7.0 Administration Guide Volume 2*.
- **ms_resource**, which is the name of the resource. Since the requested resource type for this particular example is HTTP, it is legal to prepend a host name and port number to the resource name, as in the following:

  ```
  //Example.com:80/secrets/index.html
  ```

- **ms_method**, which is the type of operation to be performed against the resource. When the resource type is HTTP, the possible operations are GET and POST. For EJB-type resources, the operation must be EXECUTE. For

custom resource types, you define the permitted operations when you set up the resource type in the Access System Console. For more information on defining resource types and protecting resources with policy domains, see the *NetPoint 7.0 Administration Guide Volume 2*.

```
ObResourceRequest rrq = new ObResourceRequest(ms_protocol,
     ms_resource, ms_method);
```

Determine whether the requested resource "rrq" is protected by an authentication scheme.

```
if (rrq.isProtected()) {
```

If the resource is protected, report that fact.

```
   System.out.println("Resource is protected.");
```

Use the ObAuthenticationScheme constructor to create an authorization scheme object named "authnScheme." Specify the resource request "rrq" so that ObAuthentication Scheme checks for the specific authorization scheme associated with that particular resource.

```
   ObAuthenticationScheme authnScheme =new ObAuthenticationScheme(rrq);
```

Determine if the authorization scheme is FORM-based.

```
   if (authnScheme.isForm()) {
```

If the authorization scheme does use HTTP FORM as the challenge method, report that fact, then create a hashtable named "creds" to hold the name:value pairs representing the user name (userid) and the user password (password). Read the values for "ms_login" and "ms_passwd" into the hashtable.

```
   System.out.println("Form Authentication Scheme.");
   Hashtable creds = new Hashtable();
   creds.put("userid", ms_login);
   creds.put("password", ms_passwd);
```

Using the ObUserSession constructor, create a user session object named "session." Specify the resource request as "rrq" and the authentication scheme as "creds" so that ObUserSession can return the new structure with state information as to whether the authentication attempt has succeeded.

```
   ObUserSession session = new ObUserSession(rrq, creds);
```

Invoke the "getStatus" method on the ObUserSession state information to determine if the user is now successfully logged in (authenticated).

```
   if (session.getStatus() == ObUserSession.LOGGEDIN) {
```

If the user is authenticated, determine if the user is authorized to access the resource specified through the resource request structure "rrq."

```
      if (session.isAuthorized(rrq)) {
         System.out.println(
               "User is logged in " +
```

```
                    "and authorized for the request " +
```

Determine the authorization level returned by the “getLevel” method for the user session named “session.”

```
                    "at level " + session.getLevel());
```

If the user is not authorized for the resource specified in “rrq,” then report that the user is authenticated but not authorized to access the requested resource.

```
                } else {
                   System.out.println("User is logged in but NOT authorized");
```

If the user is not authenticated, report that fact. (A real world application might give the user additional chances to authenticate).

```
             } else {
                System.out.println("User is NOT logged in");
```

If the authentication scheme does not use an HTTP FORM-based challenge method, report that fact. At this point, a real-world application might branch to facilitate whatever other challenge method the authorization scheme specifies, such as “basic” (which requires only userid and password), “certificate” (SSL or TLS over HTTPS), or “secure” (HTTPS through a redirection URL). For more information about challenge Methods and configuring user authentication, see the *NetPoint 7.0 Administration Guide Volume 2*.

```
          } else {
             System.out.println("non-Form Authentication Scheme.");
          }
```

If the resource is not protected, report that fact. (By implication, the user gains access to the requested resource, because the AccessGate makes no further attempt to protect the resource).

```
       } else {
          System.out.println("Resource is NOT protected.");
       }
    }
```

If an error occurs anywhere within the preceding try block, get the associated text message “oe” and report it.

```
    catch (ObAccessException oe) {
       System.out.println(
          "NetPoint Access Exception: " + oe.getMessage());
    }
```

Now that the program is finished calling the Access Server, shut down the API, thus releasing any memory the API might have maintained between calls.

```
    ObConfig.shutdown();
  }
}
```

Exit the program. You don't have to deallocate the memory used by the structures created by this application because Java Garbage Collection automatically "cleans up" unused structures when it determines that they are no longer needed.

Example: access_test_c.cpp

This sample demonstrates the use of C-language "pseudo classes" to implement a simple AccessGate. The "member functions" of these classes are really wrapped pointers that call C++ code.

Exception handling is performed by a callback function that is registered with the SDK before it is initialized. This error handling function is called from within SDK methods when an error condition needs to be reported. In this example, the error handler simply prints out the error message associated with the error code returned, then shuts down the program.

The complete listing for access_test_c.exe appears immediately below. You can cut-and-paste the code into a text file with the ".cpp" file name extension and then generate executable code using a compiler appropriate for C-language programs on the server platform where your AccessGate will reside. Annotations for this code sample begin on page 293.

Listing 30 access_test_c.cpp

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <obaccess_api_c.h>

void myExceptionHandler(ObAccessExceptionCode_t code) {
   printf("EXCEPTION: %s\n", ObAccessException_getCodeString(code));
   exit(1);
}

int main(int argc, char *argv[]) {
   const char *userid, *password, *method, *url, *location;
   ObResourceRequest_t res;
   ObAuthnScheme_t authnScheme;
   ObMap_t credentials;
   ObUserSession_t user;
   ObMap_t actions;
   ObMap_t parameters;
   ObMap_t requiredParameters;
   ObMapIterator_t iter;
   const char **actionTypes;
   const char *name, *val;
   int i;
   if (argc < 5 || argc > 7) {
      printf("EXPECTED: userid password HTTP-method URL(without http:)
           [client-location [authz-parameters]]\n");
```

```
   return 1;
}
userid   = argv[1];
password = argv[2];
method   = argv[3];
url      = argv[4];
location = argc >= 6 ? argv[5] : NULL;
if (argc == 7) {
   parameters = ObMap_new();
   for (name = strtok(argv[6], "="); name != NULL; name = strtok(NULL, "=")) {
      val = strtok(NULL, "&");
      ObMap_put(parameters, name, val);
   }
} else {
   parameters = NULL;
}
ObAccessException_setHandler(myExceptionHandler);
ObConfig_initialize(NULL);
res = ObResourceRequest_new("http", url, method, NULL);
if (ObResource_isProtected(res)) {
   authnScheme = ObAuthn_new(res);
   if (ObAuthn_isBasic(authnScheme)) {
      credentials = ObMap_new();
      ObMap_put(credentials, "userid",   userid);
      ObMap_put(credentials, "password", password);
      user = ObUserSession_authenticate(res, credentials, NULL);
      if (ObUser_getStatus(user) == ObUser_LOGGEDIN) {
         ObUser_setLocation(user, location);
         if (parameters != NULL
                  ? ObUser_isAuthorizedWithParameters(user, res, parameters)
                  : ObUser_isAuthorized(user, res)) {
            printf("GRANTED\n");
         } else {
            printf("DENIED\n");
            printf("ERROR: %s\n", ObUser_getErrorMessage(user));
            if (ObUser_getError(user) == ObUser_ERR_NEED_MORE_DATA) {
               requiredParameters = ObResource_getAuthorizationParameters(res);
               printf("REQUIRED PARAMETERS:");
               iter = ObMapIterator_new(requiredParameters);
               while (ObMapIterator_hasMore(iter)) {
                  ObMapIterator_next(iter, &name, &val);
                  printf(" ");
                  printf(name);
               }
               printf("\n");
               ObMapIterator_free(&iter);
               ObMap_free(&requiredParameters);
            }
         }
      if (parameters != NULL) ObMap_free(&parameters);
      printf("ACTIONS:");
      actionTypes = ObUser_getActionTypes(user);
      for (i = 0; actionTypes[i] != NULL; i++) {
```

```
            actions = ObUser_getActions(user, actionTypes[i]);
            iter    = ObMapIterator_new(actions);
            while (ObMapIterator_hasMore(iter)) {
               printf("\n");
               ObMapIterator_next(iter, &name, &val);
               printf("%s: %s=%s", actionTypes[i], name, val);
            }
            ObMapIterator_free(&iter);
         }
         ObUser_logoff(user);
         } else {
            const char *errmsg;
            errmsg = ObUser_getErrorMessage(user);
            printf("LOGIN FAILED: %s", errmsg);
         }
         ObUser_free(&user);
         ObMap_free(&credentials);
      } else {
         printf("RESOURCE SCHEME NOT BASIC");
      }
      ObAuthn_free(&authnScheme);
   } else {
      printf("NOT PROTECTED");
   }
   ObResource_free(&res);
   ObConfig_shutdown();
   return 0;
}
```

Annotated Code

Import three standard C libraries to support input/output, string manipulation, and other basic functionality.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

Import the C implementation of the Access Server API.

```
#include <obaccess_api_c.h>
```

Set up an exception handler, which responds to an error by reporting the associated error message, then terminating the program.

```
void myExceptionHandler(ObAccessExceptionCode_t code) {
   printf("EXCEPTION: %s\n", ObAccessException_getCodeString(code));
   exit(1);
}
```

Declare the requisite variables and constants for the main method. “Argc” represents the total number of strings in the space-delimited array “argv.” The first string in “argv” is always the name of the program, “access_test_c.exe.”

```
int main(int argc, char *argv[]) {
   const char *userid, *password, *method, *url, *location;
   ObResourceRequest_t res;
   ObAuthnScheme_t authnScheme;
   ObMap_t credentials;
   ObUserSession_t user;
   ObMap_t actions;
   ObMap_t parameters;
   ObMap_t requiredParameters;
   ObMapIterator_t iter;
   const char **actionTypes;
   const char *name, *val;
   int i;
```

If the command-line input contains fewer that 5 or more than 7 strings, tell the user what information needs to be entered from the command line, and in what order.

```
   if (argc < 5 || argc > 7) {
      printf("EXPECTED: userid password HTTP-method URL(without http:)
            [client-location [authz-parameters]]\n");
      return 1;
   }
```

If the command-line input contains 5 to 7 strings, assign the second through fifth strings (argv[1]-arg[4]) to the variables “userid,” “password,” “method,” and “url,” respectively.

```
   userid   = argv[1];
   password = argv[2];
   method   = argv[3];
   url      = argv[4];
```

If six or more strings exist in the array received from command-line input, assign the sixth argument to the variable “location.” Otherwise, set “location” to NULL.

```
   location = argc >= 6 ? argv[5] : NULL;
```

If the command-line input array contains exactly 7 strings, create a new ObMap list structure and name it “parameters.”

```
   if (argc == 7) {
      parameters = ObMap_new();
```

By convention, the seventh string (argv[6]) takes the form “n1=v1&n2=v2...” We invoke the “strtok” method to break this string into name:value token pairs and read them into the list “parameters.” The opening delimiter for each token is always NULL, because “strtok” considers the beginning of any string it parses to be NULL. Furthermore, after “strtok” finds the concluding delimiter for a token and returns the characters that compose the token, it sets everything prior to and including the concluding delimiter to NULL. Thus, the opening delimiter for every token is always NULL.

The concluding delimiter, on the other hand, changes from "=" to "&" and back again as "strtok" parses the name and value for successive parameters in the string.

```
   for (name = strtok(argv[6], "="); name != NULL; name = strtok(NULL, "=")) {
      val = strtok(NULL, "&");
      ObMap_put(parameters, name, val);
   }
```

If only 5 or 6 arguments exist in "argv," set "parameters" to NULL.

```
} else {
   parameters = NULL;
}
```

Register the callback function "myExceptionHandler" with the Access Server SDK.

```
ObAccessException_setHandler(myExceptionHandler);
```

Initialize the AccessGate without specifying the directory in which the Access Server API SDK is installed. (Since no location is specified here, the operating system uses the value stored in the environment variable OBACCESS_INSTALL_DIR.)

```
ObConfig_initialize(NULL);
```

Create an ObResourceRequest structure that specifies the following:

- The resource type is HTTP
- The target resource is the value stored in "url"
- The operation to be performed against the resource is the value stored in "method"
- The parameters required for authorization are NULL

```
res = ObResourceRequest_new("http", url, method, NULL);
```

If the requested resource is protected, create an ObAuthn structure named "authnScheme" to return information on the specific authorization scheme used to protect the resource.

```
if (ObResource_isProtected(res)) {
   authnScheme = ObAuthn_new(res);
```

If the authorization scheme is "basic," create an ObMap structure and read into it the values for "userid" and "password" that represent the user credentials.

```
   if (ObAuthn_isBasic(authnScheme)) {
      credentials = ObMap_new();
      ObMap_put(credentials, "userid",   userid);
      ObMap_put(credentials, "password", password);
```

Invoke the "ObUserSession_authenticate" method for the specified resource request and the supplied userid and password.

```
user = ObUserSession_authenticate(res, credentials, NULL);
```

If the user is logged in, which is to say, the user has authenticated successfully, assign the value stored in "location" as the IP address of the user's machine.

```
if (ObUser_getStatus(user) == ObUser_LOGGEDIN) {
   ObUser_setLocation(user, location);
```

If the structure "parameters" is not empty, then determine whether the user is authorized to access the target resource under the parameters specified by "parameters." Otherwise, determine whether the user is authorized to access the target resource without any parameters attached.

```
   if (parameters != NULL
         ? ObUser_isAuthorizedWithParameters(user, res, parameters)
         : ObUser_isAuthorized(user, res)) {
```

If the user is authorized to access the target resource, report that fact.

```
      printf("GRANTED\n");
```

Otherwise, report that the request has been denied, and report the associated error message as well.

```
   } else {
      printf("DENIED\n");
      printf("ERROR: %s\n", ObUser_getErrorMessage(user));
```

If the error code returned is ObUser_ERR_NEED_MORE_DATA, report the names of all the parameters needed for authorization. Do this by creating an ObMapIterator structure named "requiredParameters" and then reporting the names of all the required parameters (but not the corresponding values that the user must supply!)

Note: The access policy for the resource requires authorization parameters that were not supplied in the original ObUser_isAuthorized call. This happens when the authorization rule for the policy uses an authorization scheme with an authorization plug-in that requires parameters. See "Access Management API" on page 403.

```
      if (ObUser_getError(user) == ObUser_ERR_NEED_MORE_DATA) {
         requiredParameters = ObResource_getAuthorizationParameters(res);
         printf("REQUIRED PARAMETERS:");
         iter = ObMapIterator_new(requiredParameters);
         while (ObMapIterator_hasMore(iter)) {
            ObMapIterator_next(iter, &name, &val);
            printf(" ");
            printf(name);
         }
```

Clean up by destroying both the "requiredParameters" structure and the iterator "iter," which is used to extract name strings from "requiredParameters."

```
         printf("\n");
         ObMapIterator_free(&iter);
```

```
            ObMap_free(&requiredParameters);
         }
      }
```

If “parameters” is not empty, deallocate the memory used by the structure.

```
if (parameters != NULL) ObMap_free(&parameters);
```

Report all the actions defined by the authentication and authorization rules for the policy that applies to the resource. These can be any sequence of the form “type:name:value:value:type.” ObUser_getActionTypes returns an array of the action types (such as headerVar) present in the sequence of actions. ObUser_getActions returns an ObMap structure of the actions for each action type in turn. “iter” steps through each action in each ObMap structure.

```
printf("ACTIONS:");
actionTypes = ObUser_getActionTypes(user);
for (i = 0; actionTypes[i] != NULL; i++) {
   actions = ObUser_getActions(user, actionTypes[i]);
   iter    = ObMapIterator_new(actions);
   while (ObMapIterator_hasMore(iter)) {
      printf("\n");
      ObMapIterator_next(iter, &name, &val);
      printf("%s: %s=%s", actionTypes[i], name, val);
   }
```

Destroy the string “iter” used to extract the information from the ObMapIterator structure “actions.”

```
   ObMapIterator_free(&iter);
```

Set the local ObUserSession structure to the “logged off” state.

Note: To prevent residual session tokens (such as those stored in cookies on the user’s browser) from being used to recreate the session, you must explicitly reset them using the logged off user session.

```
}
ObUser_logoff(user);
```

Otherwise, report that authentication has failed, and report the associated error message as well.

```
} else {
   const char *errmsg;
   errmsg = ObUser_getErrorMessage(user);
   printf("LOGIN FAILED: %s", errmsg);
}
```

Clean up by deallocating the memory for the ObUser and ObMap structures named “user” and “credentials,” respectively.

```
ObUser_free(&user);
ObMap_free(&credentials);
```

If the authentication scheme is not basic, report that fact.

```
      } else {
         printf("RESOURCE SCHEME NOT BASIC");
      }
```

Clean up by deallocating the memory used by the ObAuthn structure named "authnScheme."

```
      ObAuthn_free(&authnScheme);
```

If the requested resource is not protected, report that fact.

```
   } else {
      printf("NOT PROTECTED");
   }
```

Clean up by deallocating the memory used by the ObResourceRequest structure named "res." Then shutdown the AccessGate, returning "0" to indicate successful completion.

```
   ObResource_free(&res);
   ObConfig_shutdown();
   return 0;
}
```

Exit the program.

Example: Java Login Servlet

This example follows the basic pattern of API calls that define an AccessGate, as described in the JAccessGate example. However, this example is implemented as a Java servlet running within a Web server, or even an application server. In this environment, the AccessGate servlet has an opportunity to play an even more important role for the user of a Web application. By storing a NetPoint session token in the user's HTTP session, the servlet can facilitate single sign-on for the user. In other words, the authenticated Access Server session information that the first request establishes is not discarded after one authorization check. Instead, the stored session token is made available to server-side application components such as beans and other servlets, so that they do not need to interrupt the user again and again to request the same credentials. For a detailed discussion of NetPoint session tokens, ObSSO cookies, and configuring Single Sign-on, see the *NetPoint 7.0 Administration Guide Volume 2*.

This sample login servlet accepts userid/password parameters from a form on a custom login page, and attempts to log the user into NetPoint. On successful logon, the servlet stores a NetPoint session token in the ObUserSession object. This enables subsequent requests in the same HTTP session to bypass the authentication step (providing the subsequent requests use the same authentication scheme as the original request), thereby achieving Single Sign-on (SSO).

A complete listing for the Java login servlet appears immediately below. This code can provide the basis for a plug-in to a web server or application server. An annotated version of this code begins on page 300.

Listing 31 Java LoginServlet Example.

```
package obaccess;

import java.io.*;
import java.util.*;
import java.text.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.IOException;
import java.util.*;
import com.oblix.access.*;

public class LoginServlet extends HttpServlet {

   public void init(ServletConfig config) throws ServletException {
      try {
         ObConfig.initialize("install directory of oblix access server sdk");
      } catch (ObAccessException oe) {
         oe.printStackTrace();
      }
   }

   public void service(HttpServletRequest request, HttpServletResponse response)
           throws IOException, ServletException {
      ObAuthenticationScheme authnScheme = null;
      ObUserSession user = null;
      ObResourceRequest resource = null;
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();
      out.println("<HTML>");
      out.println("<HEAD><TITLE>LoginServlet: Error Page</TITLE></HEAD>");
      out.println("<BODY>");
      HttpSession session = request.getSession( false);
      String requestedPage = request.getParameter(Constants.REQUEST);
      String reqMethod = request.getMethod();
      Hashtable cred = new Hashtable();
      try {
         if (requestedPage == null) {
            out.println("<p>REQUESTED PAGE NOT SPECIFIED\n");
            out.println("</BODY></HTML>");
            return;
         }
         resource = new ObResourceRequest("http", requestedPage, "GET");
         if (resource.isProtected()) {
            authnScheme = new ObAuthenticationScheme(resource);
            if (authnScheme.isBasic()) {
               if (session == null) {
                  String sUserName = request.getParameter( Constants.USERNAME);
```

```
                String sPassword = request.getParameter( Constants.PASSWORD);
                if (sUserName != null) {
                   cred.put("userid", sUserName);
                   cred.put("password", sPassword);
                   user = new ObUserSession(resource, cred);
                   if (user.getStatus() == ObUserSession.LOGGEDIN) {
                      if (user.isAuthorized(resource)) {
                         session = request.getSession( true);
                         session.putValue( Constants.OBUSER, user);
                         response.sendRedirect( requestedPage );
                      } else {
                         out.println("<p>User " + sUserName + " not
                               authorized for " + requestedPage + "\n");
                      }
                   } else {
                      out.println("<p>User" + sUserName + "NOT LOGGED IN\n");
                   }
                } else {
                   out.println("<p>USERNAME PARAM REQUIRED\n");
                }
             } else {
                user = (ObUserSession)session.getValue(Constants.OBUSER);
                if (user.getStatus() == ObUserSession.LOGGEDIN) {
                   out.println("<p>User " + user.getUserIdentity() + " already
                         LOGGEDIN\n");
                }
             }
          } else {
             out.println("<p>Resource Page" + requestedPage + " is not
                   protected with BASIC\n");
          }
       } else {
          out.println("<p>Page " + requestedPage + " is not protected\n");
       }
    } catch (ObAccessException oe) {
          oe.printStackTrace();
    }
   out.println("</BODY></HTML>");
   }
}
```

Annotated Code

All the classes defined in this listing belong to the package named “obaccess.”

```
package obaccess;
```

Import three standard Java packages to support input/output, text manipulation, and basic functionality.

```
import java.io.*;
import java.util.*;
```

```
import java.text.*;
```

Import two packages of Java extensions to provide servlet-related functionality.

```
import javax.servlet.*;
import javax.servlet.http.*;
```

Import a standard Java package to handle exceptions.

```
import java.io.IOException;
```

Import the package “com.oblix.access.jar,” which is the Java implementation of the Access Server API.

```
import com.oblix.access.*;
```

This servlet, which builds on the functionality of the generic HttpServlet supported by the Java Enterprise Edition, is named “LoginServlet.”

```
public class LoginServlet extends HttpServlet {
```

The “init” method is called once by the servlet engine to initialize the AccessGate. In the case of initialization failure, report that fact, along with the appropriate error message.

```
   public void init() {
        ObConfig.initialize(“install directory of the oblix access server sdk”);
      } catch (ObAccessException oe) {
           oe.printStackTrace();
      }
   }
```

Invoke the “javax.servlet.service” method to process the user’s resource request.

```
   public void service(HttpServletRequest request, HttpServletResponse response)
           throws IOException, ServletException {
```

Initialize as NULL the variables that store the ObAccess structures used to process the resource request, then set the response type used by this application to “text/html.”

```
      ObAuthenticationScheme authnScheme = null;
      ObUserSession user = null;
      ObResourceRequest resource = null;
      response.setContentType("text/html");
```

Open an output stream titled “LoginServlet: Error Page” and direct it to the user’s browser.

```
      PrintWriter out = response.getWriter();
      out.println("<HTML>");
      out.println("<HEAD><TITLE>LoginServlet: Error Page</TITLE></HEAD>");
      out.println("<BODY>");
```

Determine if a session already exists for this user. Invoke the "getSession" method with "false" as a parameter, so the value of the existing servlet session (and not the ObUserSession) will be returned if it is present; otherwise, NULL will be returned.

```
HttpSession session = request.getSession(false);
```

Retrieve the name of the target resource, assign it to the variable "requestedPage," then retrieve the name of the HTTP method (such as GET, POST, or PUT) with which the request was made and assign it to the variable "reqMethod."

```
String requestedPage = request.getParameter(Constants.REQUEST);
String reqMethod = request.getMethod();
```

Create a hashtable named "cred" to hold the user's credentials.

```
Hashtable cred = new Hashtable();
```

If the variable "requestedPage" is returned empty, report that the name of the target resource has not been properly specified, then terminate the servlet.

```
try {
   if (requestedPage == null) {
      out.println("<p>REQUESTED PAGE NOT SPECIFIED\n");
      out.println("</BODY></HTML>");
      return;
   }
```

If the name of the requested page is returned, create an ObResourceRequest structure and set the following:

- The resource type is HTTP
- The HTTP method is GET
- "resource" is the value stored by the variable "requestedPage"

```
   resource = new ObResourceRequest("http", requestedPage, "GET");
```

If the target resource is protected, create an ObAuthenticationScheme structure for the resource request and name it "authnScheme."

```
   if (resource.isProtected()) {
      authnScheme = new ObAuthenticationScheme(resource);
```

If the authentication scheme associated with the target resource is HTTP "basic" and no user session currently exists, invoke javax.servlet.servletrequest. getParameter to return the user's credentials (user name and password) and assign them to the variables sUserName and sPassword, respectively.

Note: For the authnScheme.isBasic call in the following statement to work properly, the username and password must be included in the query string of the user's HTTP request, as in the following:

```
http://host.example.com/resource?username=bob&userpassword=
bobspassword
```

where *resource* is the resource being requested, *bob* is the user making the request, and *bobspassword* is the user's password.

Note: If you substitute authnScheme.isForm for authnScheme.isBasic, you need to write additional code to implement the following steps.

Additional Code for authnScheme.isForm will

1. Process the original request and determine that form-based login is required.
2. Send a 302 redirect response for the login form and also save the original resource information in the HTTP session.
3. Authenticate the user by processing the posted form data with the user's name and password.
4. Retrieve the original resource from the HTTP resource and send a 302 redirect response for the original resource.
5. Process the original request once again, this time using the ObUserSession stored int he HTTP session.

```
if (authnScheme.isBasic()) {
   if (session == null) {
      String sUserName = request.getParameter(Constants.USERNAME);
      String sPassword = request.getParameter(Constants.PASSWORD);
```

If the user name exists, read it, along with the associated password, into the hashtable named "cred."

```
      if (sUserName != null) {
         cred.put("userid", sUserName);
         cred.put("password", sPassword);
```

Create a user session based on the information in the ObResourceRequest structure named "resource" and the hashtable "cred."

```
         user = new ObUserSession(resource, cred);
```

If the status code for the user returns as LOGGEDIN, that user has authenticated successfully.

```
         if (user.getStatus() == ObUserSession.LOGGEDIN) {
```

Determine if the user is authorized to access the target resource.

```
            if (user.isAuthorized(resource)) {
```

Create a servlet user session (which is not to be confused with an ObUserSession) and add the name of the user to it.

```
               session = request.getSession( true);
               session.putValue( Constants.OBUSER, user);
```

Redirect the user's browser to the target page.

```
                response.sendRedirect(requestedPage);
```

If the user is not authorized to access the target resource, report that fact.

```
            } else {
               out.println("<p>User " + sUserName + " not authorized
                     for " + requestedPage + "\n");
            }
```

If the user is not properly authenticated, report that fact.

```
         } else {
            out.println("<p>User" + sUserName + "NOT LOGGED IN\n");
         }
```

If the user name has not been supplied, report that fact.

```
      } else {
      out.println("<p>USERNAME PARAM REQUIRED\n");
      }
```

If a session already exists, retrieve “OBUSER” and assign it to the session variable “user.”

```
      } else {
         user = (ObUserSession)session.getValue(Constants.OBUSER);
```

If the user is logged in, which is to say, the user has authenticated successfully, report that fact along with the user’s name.

```
         if (user.getStatus() == ObUserSession.LOGGEDIN) {
            out.println("<p>User " + user.getUserIdentity() + " already
                  LOGGEDIN\n");
         }
      }
```

If the target resource is not protected by a “basic” authentication scheme, report that fact.

```
   } else {
      out.println("<p>Resource Page" + requestedPage + " is not protected
            with BASIC\n");
   }
```

If the target resource is not protected by any authentication scheme, report that fact.

```
} else {
   out.println("<p>Page " + requestedPage + " is not protected\n");
}
```

If an error occurs, report the backtrace.

```
} catch (ObAccessException oe) {
   oe.printStackTrace();
}
```

Complete the output stream to the user’s browser.

```
   out.println("</BODY></HTML>");
   }
}
```

Example: access_api_test.cs

This sample program demonstrates the C# (.NET managed code) API for the Access Server.

Process overview: Sample program

1. Initializes the system

   Note: Before you run this program on either a Windows or Unix, make sure the environment variable OBACCESS_INSTALL_DIR is defined to point to *SDK_install_dir*.

2. Creates a resource request and determines if the target resource is protected.
3. Checks whether authentication for this resource is "basic."
4. Logs in the user named *cuser10k429*, as defined in the NetPoint user directory.

   Note: Change the name of the user and other particulars to match a valid entry in your NetPoint user directory.

5. Verifies that the user is logged in.
6. Checks if the user is authorized to access the target resource.
7. Obtains the identity of the user and the location of the user's machine.

The following line represents typical command-line input for this program:

```
access_api_test //www.example.com:88/managed
```

For specifics on compiling, linking, and running this .NET program, consult the following file:

SDK_install_dir\samples\access_csharp\README.txt

A complete listing for Access_API_Test appears immediately below, with an annotated code section following on page 307.

Listing 32 Access_API_Test

```
using System;
using System.Reflection;
using System.Collections;
using Oblix.Access.Server;
using Oblix.Access.Common;
```

```
class Access_API_Test {
   public static int Main(string[] args) {
      String resourceString = "//www.oblix.com:80/managed";
      if ( args.Length > 0 )resourceString = args[0];
      Console.WriteLine("Initialize the configuration directory!");
      try {
         String config = "../../../../";
         ObConfigMgd.initialize(config);
      } catch (ObAccessExceptionMgd ex) {
         Console.WriteLine("Initialization Exception caught: " + ex.String);
         return -1;
      }
      ObDictionary parameters = new ObDictionary();
      ObResourceRequestMgd resource = new
            ObResourceRequestMgd("http",resourceString,"GET",parameters);
      if ( resource.IsProtected == true ) {
         Console.WriteLine("Resource " + resourceString + " is protected ..." );
         try {
            ObAuthenticationSchemeMgd authnScheme = new
                  ObAuthenticationSchemeMgd(resource);
            if ( authnScheme.IsBasic ) {
               Console.WriteLine("Authentication is basic" );
               ObDictionary credentials = new ObDictionary();
               credentials.Add("userid","cuser10k429");
               credentials.Add("password","oblix");
               ObUserSessionMgd user = new ObUserSessionMgd(resource,credentials);
               ObUserStatusMgd status = user.Status;
               if ( !status.IsLoggedIn ) {
                  Console.WriteLine("User is not logged in");
                  return -1;
               }
               user.Location = "127.0.0.1";
               Console.WriteLine("User: " + user.UserIdentity + " is logged
                     in...");
               Console.WriteLine("User location is: " + user.Location);
               if ( user.IsAuthorized(resource) ) {
                  Console.WriteLine("User is authorized");
               } else {
                  Console.WriteLine("User is not authorized");
               }
            } else {
               Console.WriteLine("Authentication is not basic" );
            }
         } catch (ObAccessExceptionMgd ex) {
            Console.WriteLine("Access Exception caught: " + ex.String);
            return -1;
         }
      } else {
         Console.WriteLine("Resource is NOT protected ... " );
      }
      return 1;
   }
```

```
}
```

Annotated Code

Import three .NET framework libraries to provide type management for loaded methods, support for lists and hashtables, and basic functionality.

```
using System;
using System.Reflection;
using System.Collections;
```

Import the C# implementation of the Access Server API, which resides in a main library as well as a shared library for code used in common with the Access Management API.

```
using Oblix.Access.Server;
using Oblix.Access.Common;
```

This program is named "Access_API_Test.cs."

```
class Access_API_Test {
   public static int Main(string[] args) {
```

This program does not retrieve user input from forms or session tokens. For convenience, we simply assign sample values to the parameters associated with the methods being demonstrated.

```
      String resourceString = "//www.example.com:80/managed";
```

The C# application expects a maximum of one argument in the command line. If the array "args" contains one or more arguments, assign the first string in "args" to the variable "resourceString," thus replacing the URL string we just assigned to it. If there are no arguments, go ahead and use the string we assigned to "resourceString" in the previous statement.

```
      if ( args.Length > 0 )resourceString = args[0];
      Console.WriteLine("Initialize the configuration directory!");
      try {
         String config = "../../../../";
         ObConfigMgd.initialize(config);
```

If initialization fails, report that fact, along with the associated error message, then return a value of "-1," effectively terminating the program. (In this particular C# program, execution failure returns "-1." By contrast, the C++ sample "access_test_c" returns a "1" when execution fails and "0" when execution completes successfully.)

```
      } catch (ObAccessExceptionMgd ex) {
         Console.WriteLine("Initialization Exception caught: " + ex.String);
         return -1;
      }
```

Create an ObDictionary structure named “parameters” to hold the user’s credentials.

```
ObDictionary parameters = new ObDictionary();
```

Create a resource request object, specifying parameters to accomplish the following:

- Set HTTP as the resource type
- Designate the URL previously assigned to “resourceString” as the target resource
- Specify GET as the action to be performed against the resource
- Store the parameters required for authentication in the ObDictionary structure “parameters”

```
ObResourceRequestMgd resource = new
     ObResourceRequestMgd("http",resourceString,"GET",parameters);
```

Determine whether the target resource is protected, and if it is, report that fact.

```
if ( resource.IsProtected == true ) {
   Console.WriteLine("Resource " + resourceString + " is protected ..." );
```

Create an object to return information about the authentication scheme associated with the requested resource.

```
   try {
      ObAuthenticationSchemeMgd authnScheme = new
           ObAuthenticationSchemeMgd(resource);
```

If the authentication scheme is “basic,” report that fact.

```
      if ( authnScheme.IsBasic ) {
         Console.WriteLine("Authentication is basic" );
```

Create an ObDictionary structure named “credentials” to store the user’s login credentials. In this test application, the userid and password are supplied within the source code; in a real-world application, these would be retrieved from keyboard input or a session token.

```
ObDictionary credentials = new ObDictionary();
credentials.Add("userid","cuser10k429");
credentials.Add("password","oblix");
```

Create a ObUserSessionMgd structure to store information about the current user, the resource requested, and the authentication scheme associated with that resource.

```
ObUserSessionMgd user = new ObUserSessionMgd(resource,credentials);
ObUserStatusMgd status = user.Status;
```

If the user is not logged in (in other words, the user has not authenticated successfully), report that fact and return -1, effectively terminating the program.

```
            if (!status.IsLoggedIn) {
               Console.WriteLine("User is not logged in");
               return -1;
            }
```

Set the IP of the user's machine to "127.0.0.1." If this were a real-life application, we would retrieve this location from keyboard input or the session token.

```
            user.Location = "127.0.0.1";
```

Report that the user is logged in, then report the IP of the user's browser.

```
            Console.WriteLine("User: " + user.UserIdentity + " is logged
                 in...");
            Console.WriteLine("User location is: " + user.Location);
```

Report whether or not the user is authorized to access the specified resource.

```
            if ( user.IsAuthorized(resource) ) {
               Console.WriteLine("User is authorized");
            } else {
               Console.WriteLine("User is not authorized");
            }
```

If the authentication scheme associated with the requested resource is not basic, report that fact.

```
         } else {
            Console.WriteLine("Authentication is not basic" );
         }
```

If an execution error occurs, report the associated error message, then return "-1," effectively terminating the program.

```
      } catch (ObAccessExceptionMgd ex) {
         Console.WriteLine("Access Exception caught: " + ex.String);
         return -1;
      }
```

If the resource is not protected, report that fact and return "1," thus terminating the program after indicating successful execution.

```
   } else {
      Console.WriteLine("Resource is NOT protected ... " );
   }
   return 1;
  }
}
```

Example: access_test_java.java

Building on the basic pattern established in the sample application "JAccessGate.java," the following sample program invokes several additional Access Server methods. For instance, it inspects the NetPoint session object to determine which actions are currently configured in the policy rules associated with the current authentication scheme.

For this demonstration to take place, you must configure some actions through the Access System console prior to running the application. For details about authentication action and configuring user authentication, see the *NetPoint 7.0 Administration Guide Volume 2*. The complete listing for this sample application appears immediately below and also in the file "access_test_java.java," which is located in the directory *SDK_install_dir*\Samples. An annotated version of the code begins on page 313.

Listing 33 access_test_java.java

```
import java.util.*;
import java.util.Enumeration;
import java.util.StringTokenizer;
import com.oblix.access.*;

public class access_test_java {

   public static void main(String[] arg) {
      String userid, password, method, url, configDir, type, location;
      ObResourceRequest res;
      Hashtable parameters = null;
      Hashtable cred = new Hashtable();
      if (arg.length < 5) {
         System.out.println("Usage: EXPECTED: userid password Type HTTP-method
               URL [Installdir [authz-parameters] [location]]]");
         return;
      } else {
         userid= arg[0];
         password= arg[1];
         type    = arg[2];
         method= arg[3];
         url     = arg[4];
      }
      if (arg.length >= 6) {
         configDir = arg[5];
      } else {
         configDir = null;
      }
      if (arg.length >= 7 && arg[6] != null) {
         parameters = new Hashtable();
         StringTokenizer tok1 = new StringTokenizer(arg[6], "&");
         while (tok1.hasMoreTokens()) {
            String nameValue = tok1.nextToken();
```

```
            StringTokenizer tok2 = new StringTokenizer(nameValue, "=");
            String name = tok2.nextToken();
            String value = tok2.hasMoreTokens() ? tok2.nextToken() : "";
            parameters.put(name, value);
         }
      }
      location = arg.length >= 8 ? arg[7] : null;
      try {
         if (configDir != null) {
            ObConfig.initialize(configDir);
         } else {
            ObConfig.initialize();
         }
      } catch (ObAccessException ie) {
         System.out.println("Access Server SDK Initialization failed");
         ie.printStackTrace();
         return;
      }
      cred.put("userid", userid);
      cred.put("password", password);
      try {
         res = new ObResourceRequest(type, url, method);
         if (res.isProtected()) {
            System.out.println("Resource " + type + ":" + url + " protected");
         } else {
            System.out.println("Resource " + type + ":" + url + " unprotected");
         }
      } catch (Throwable t) {
         t.printStackTrace();
         System.out.println("Failed to created new resource request");
         return;
      }
      ObUserSession user = null;
      try {
         user = new ObUserSession(res, cred);
      } catch (Throwable t) {
         t.printStackTrace();
         System.out.println("Failed to create new user session");
         return;
      }
      if (user.getStatus() == ObUserSession.LOGGEDIN) {
         if (location != null) user.setLocation(location);
         System.out.println("user status is " + user.getStatus());
         try {
            if (parameters != null ? user.isAuthorized(res, parameters) :
                  user.isAuthorized(res)) {
               System.out.println("Permission GRANTED");
               System.out.println("User Session Token =" +
                     user.getSessionToken());
               if (location != null) {
                  System.out.println("Location = " + user.getLocation());
               }
            } else {
```

```
            System.out.println("Permission DENIED");
            if (user.getError() == ObUserSession.ERR_NEED_MORE_DATA) {
               int nParams = res.getNumberOfAuthorizationParameters();
               System.out.print("Required Authorization Parameters (" +
                     nParams + ") :");
               Enumeration e = res.getAuthorizationParameters().keys();
               while (e.hasMoreElements()) {
                  String name = (String) e.nextElement();
                  System.out.print(" " + name);
               }
               System.out.println();
            }
         }
      } catch (ObAccessException obe) {
         System.out.println("Failed to get user authorization");
      }
   } else {
      System.out.println("user status is " + user.getStatus());
   }
   String[] actionTypes = user.getActionTypes();
   for(int i =0; i < actionTypes.length; i++) {
      Hashtable actions = user.getActions(actionTypes[i]);
      Enumeration e = actions.keys();
      int item = 0;
      System.out.println("Printing Actions for type " + actionTypes[i]);
      while(e.hasMoreElements()) {
         String name = (String)e.nextElement();
         System.out.println("Actions[" + item +"]: Name " + name + " value " +
               actions.get(name));
         item++;
      }
   }
   ObAuthenticationScheme auths;
   try {
      auths = new ObAuthenticationScheme(res);
   } catch (ObAccessException ase) {
      ase.printStackTrace();
      return;
   }
if (auths.isBasic()) {
      System.out.println("Auth scheme is Basic");
   } else {
      System.out.println("Auth scheme is NOT Basic");
   }
   try {
      ObResourceRequest resNew = (ObResourceRequest) res.clone();
      System.out.println("Clone resource Name: " + resNew.getResource());
   } catch (Exception e) {
      e.printStackTrace();
   }
   res = null;
   auths = null;
   ObConfig.shutdown();
```

```
   }
}
```

Annotated Code

Import standard Java libraries to provide basic utilities, enumeration, and token processing capabilities.

```
import java.util.*;
import java.util.Enumeration;
import java.util.StringTokenizer;
```

Import the Access Server API libraries.

```
import com.oblix.access.*;
```

This servlet is named “access_test_java.”

```
public class access_test_java {
```

Declare seven variable strings to store the values passed through the array named “arg.”

```
   public static void main(String[] arg) {
      String userid, password, method, url, configDir, type, location;
```

Set the current ObResourceRequest to “res.”

```
      ObResourceRequest res;
```

Initialize the hashtable parameters to NULL, just in case they were not already empty.

```
      Hashtable parameters = null;
```

Create a new hashtable named “cred.”

```
      Hashtable cred = new Hashtable();
```

If the array named “arg” contains less than five strings, report the expected syntax and content for command-line input, which is five mandatory arguments in the specified order, as well as the optional variables “configDir,” “authz-parameters,” and “location.”

```
      if (arg.length < 5) {
         System.out.println("Usage: EXPECTED: userid password type
            HTTP-method URL [configDir [authz-parameters] [location]]]");
```

Since fewer than five arguments were received the first time around, break out of the main method, effectively terminating program execution.

```
         return;
      } else {
```

If the array named “arg” contains five or more strings, assign the first five arguments (arg[0] through arg[4]) to the variables userid, password, type, method, and url, respectively.

```
   userid = arg[0];
   password = arg[1];
   type = arg[2];
   method = arg[3];
   url = arg[4];
}
```

If “arg” contains six or more arguments, assign the sixth string in the array to the variable “configDir.”

```
if (arg.length >= 6)
   configDir = arg[5];
```

If “arg” does not contain six or more arguments (in other words, we know it contains exactly five arguments, because we have already determined it does not contain fewer than five) then set “configDir” to NULL.

```
else
   configDir = null;
```

If arg contains at least seven strings, and arg[6] (which has been implicitly assigned to the variable “authz-parameters”) is not empty, create a new hashtable named “parameters.” The syntax for the string “authz-parameters” is: p1=v1&p2=v2&...

```
if (arg.length >= 7 && arg[6] != null) {
   parameters = new Hashtable();
```

Create a string tokenizer named tok1 and parse arg[6], using the ampersand character (&) as the delimiter. This breaks arg[6] into an array of tokens in the form p*n*=v*n*, where *n* is the sequential number of the token.

```
   StringTokenizer tok1 = new StringTokenizer(arg[6], "&");
```

For all the items in tok1, return the next token as the variable “nameValue.” In this manner, nameValue is assigned the string p*n*=v*n*, where *n* is the sequential number of the token.

```
   while (tok1.hasMoreTokens()) {
      String nameValue = tok1.nextToken();
```

Create a string tokenizer named tok2 and parse “nameValue,” using the equal character (=) as the delimiter. In this manner, p*n*=v*n* breaks down into the tokens p*n* and v*n*.

```
      StringTokenizer tok2 = new StringTokenizer(nameValue, "=");
```

Assign the first token to the variable “name.”

```
      String name = tok2.nextToken();
```

Assign the second token to “value.” If additional tokens remain in tok2, return the next token and assign it to “value;” otherwise, assign an empty string to “value.”

```
      String value = tok2.hasMoreTokens() ? tok2.nextToken() : "";
```

Insert “name” and “value” into the hashtable “parameters.”

```
      parameters.put(name, value);
   }
}
```

If there are eight or more arguments in “arg,” assign arg[7] to the variable “location;” otherwise make location empty.

```
location = arg.length >= 8 ? arg[7] : null;
```

If “configDir” is not empty, initialize the Access Server API using the current value of “configDir.”

```
   try {
      if (configDir != null)
         ObConfig.initialize(configDir);
```

Otherwise, initialize the Access Server API without specifying an explicit “configDir” location.

```
      else
         ObConfig.initialize();
   }
```

If the initialization attempt produces an error, report the appropriate error message (ie) to the standard error stream along with the backtrace.

```
catch (ObAccessException ie) {
   System.out.println("Initialize failed");
   ie.printStackTrace();
```

Break out of the main method, effectively terminating the program.

```
   return;
}
```

Read the variables userid and password into the hashtable named “cred.”

```
cred.put("userid", userid);
cred.put("password", password);
```

Create an ObResourceRequest object named “res,” which will return values for the variables type, url and method from the Access Server.

```
try {
   res = new ObResourceRequest(type, url, method);
```

Determine whether the requested resource “res” is protected and display the appropriate message.

```
   if (res.isProtected())
      System.out.println("Resource " + type ":" + url + " protected");
```

```
   else
      System.out.println("Resource " + type + ":" + url + " unprotected");
}
```

If the attempt to create the ObResourceRequest structure does not succeed, report the failure along with the error message “t.”

```
catch (Throwable t) {
   t.printStackTrace();
   System.out.println("Failed to create new resource request");
```

Break out of the main method, effectively terminating the program.

```
   return;
}
```

Set the ObUserSession parameter “user” to empty.

```
ObUserSession user = null;
```

Create a ObUserSession structure named “user” so that it will return values for the ObResourceRequest structure “res” and the ObAuthenticationScheme structure “cred.”

```
try
   user = new ObUserSession(res, cred);
```

If the attempt to create the ObUserSession structure does not succeed, then report the failure along with the error message “t.”

```
catch (Throwable t) {
   t.printStackTrace();
   System.out.println("Failed to create new user session");
```

Break out of the main method, effectively terminating the program.

```
   return;
}
```

Determine if the user is currently logged in, which is to say, authentication for this user has succeeded.

```
if (user.getStatus() == ObUserSession.LOGGEDIN) {
```

If the user is logged in, determine whether the variable “location” is not empty. If “location” is not empty, set the “location” parameter for ObConfig to the value of the variable “location,” then report that the user is logged in along with the status code returned by the Access Server.

```
   if (location != null) user.setLocation(location);
   System.out.println("user status is " + user.getStatus());
```

Check authorization. To accomplish this, determine whether “parameters” exists. If it does, determine whether the user is authorized with respect to the target resource when the parameters stored in “parameters” are attached. If “parameters” does not exist, simply determine whether the user is authorized for the target resource.

```
        try {
           if (parameters != null ? user.isAuthorized(res, parameters) :
                 user.isAuthorized(res)) {
```

If the user is authorized to access the resource when all the appropriate parameters have been specified, report that permission has been granted.

```
              System.out.println("Permission GRANTED");
```

Display also a serialized representation of the user session token.

```
              System.out.println("User Session Token =" +
user.getSessionToken());
```

If the variable location is not empty, report the location.

```
              if (location != null) {
                 System.out.println("Location = " + user.getLocation());
              }
```

If the user is not authorized to access the resource, report that permission has been denied.

```
           } else {
              System.out.println("Permission DENIED");
```

If ObUserSession returns ERR_NEED_MORE_DATA, set the variable "nParams" to the number of parameters required for authorization, then report that number to the user.

```
              if (user.getError() == ObUserSession.ERR_NEED_MORE_DATA) {
                 int nParams = res.getNumberOfAuthorizationParameters();
                 System.out.print("Required Authorization Parameters (" +
                    nParams + ") :");
```

Set "e" to the value of the "keys" parameter in the hashtable returned by the getAuthorizationParameters method for the ObResourceRequest object named "res."

```
                 Enumeration e = res.getAuthorizationParameters().keys();
```

Report the names of all the elements contained in "e."

```
                 while (e.hasMoreElements()) {
                    String name = (String) e.nextElement();
                    System.out.print(" " + name);
                 }
                 System.out.println();
              }
```

Otherwise, simply proceed to the next statement.

```
              else
           }
        }
```

In the case of an error, report that the authorization attempt failed.

```
        catch (ObAccessException obe)
```

```
      System.out.println("Failed to get user authorization");
}
```

If the user is not logged in, report the current user status.

```
else
   System.out.println("user status is " + user.getStatus());
```

Now report all the actions currently set for the current user session. Do this by creating an array named “actionTypes” from the strings returned by the getActionTypes method. Next, read each string in “actionTypes” into a hashtable named “actions.” Report the name and value of each of the keys contained in “actions.”

```
String[] actionTypes = user.getActionTypes();
for(int i =0; actionTypes[i] != null; i++){
   Hashtable actions = user.getActions(actionTypes[i]);
   Enumeration e = actions.keys();
   int item = 0;
   System.out.println("Printing Actions for type " + actionTypes[i]);
   while(e.hasMoreElements()) {
      String name = (String)e.nextElement();
      System.out.println("Actions[" + item +"]: Name " + name + " value " +
         actions.get(name));
      item++;
   }
}
```

Attempt to create an ObAuthenticationScheme object named “auths” for the ObResourceRequest object “res.”

```
ObAuthenticationScheme auths;
try
   auths = new ObAuthenticationScheme(res);
```

If the ObAuthenticationScheme creation attempt is unsuccessful, report the failure along with the error message “ase.”

```
catch (ObAccessException ase) {
   ase.printStackTrace();
```

Break out of the main method, effectively terminating the program.

```
   return;
}
```

Determine if the authorization scheme is basic.

```
if (auths.isBasic())
```

If it is, report the fact.

```
   System.out.println("Auth scheme is Basic");
```

It it is not basic, report the fact.

```
else
   System.out.println("Auth scheme is NOT Basic");
```

Use the copy constructor to create a new ObResourceRequest object named "resNEW" from the original object "res."

```
      try {
         ObResourceRequest resNew = (ObResourceRequest) res.clone();
```

Report the name of the newly cloned object.

```
         System.out.println("Clone resource Name: " + resNew.getResource());
```

If the ObResourceRequest object cannot be cloned for any reason, report the failure along with the associated backtrace.

```
      }
      catch (Exception e) {
         e.printStackTrace();
      }
```

Set the ObResourceRequest object "res" and the ObAuthenticationScheme object "auths" to NULL, then disconnect the Access Server API.

```
      res = null;
      auths = null;
      ObConfig.shutdown();
   }
}
```

Example: access_test_cplus.cpp

Access_test_cplus.cpp, which is written with the C++ implementation of the Access Server API, demonstrates a wide range of possible AccessGate features. The code sections roughly parallel those of the sample program access_test_java.java, but the syntax, I/O, and exception handling conform to C++.

The following subsections present an overview of the application, followed by the complete, unannotated listing, which can be copied into a C++ development environment for editing and compilation. Line-by-line annotations of the code begin on page 325.

The program calls Access Server API methods to set up a user session through either a session token or a userid and password. The program then determines whether to grant the user access to the requested HTTP resource.

There are two modes of operation:

- **Command Line**—Command-line arguments are used for a single transaction.
- **Interactive**—No command-line arguments are used; instead, multiple transactions are read from stdin.

The possible transaction types are:

- **Userid/password/method/URL**—Uses a specified challenge method to authenticate the user and to determine whether to grant access to the requested URL.
- **Token/method/URL**—Retrieves user information from a serial token, then employs a specified challenge method to determine whether to grant access to the requested URL.
- **Method/URL (Interactive mode only)**—Retrieves user information from a previous login and employs a specified challenge method to determine whether to grant access to the requested URL.

The URL to the requested resource takes the following form:

```
[resourceType:][//host[:port]]/resource[?p1=v1&p2=v2...]
```

The default resourceType is HTTP.

Examples

```
J.Smith 84CharingXRd GET /example/resource:
```

This command-line input logs in the user J. Smith with the password 84CharingXRd and checks GET access for the url "/example/resource."

```
GET /a/b.cgi?a=1&b=2:
```

This command-line input, which bases the transaction on an existing session, checks GET access to determine authorization for the url /a/b.cgi with parameters a=1 and b=2.

Listing 34 Access_test_cplus.cpp

```
#ifdef _WIN32
#pragma warning(disable : 4995)
#endif
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <iostream.h>
#ifdef _WIN32
#   include <sys/timeb.h>
#else
#   include <time.h>
#endif
#include <obaccess_api_cplus.h>
const char *sessionToken = NULL;

void parseUrl(char *urlCopy, const char **resourceType, const char **resource,
      ObMap &parameters) {
   char *colon   = strstr(urlCopy, ":");
```

```
   char *slashes = strstr(urlCopy, "//");
   if (colon != NULL && (slashes == NULL || colon < slashes)) {
      *resourceType = strtok(urlCopy, ":");
      *resource     = strtok(NULL,    "?");
   } else {
      *resourceType = "http";
      *resource     = strtok(urlCopy, "?");
   }
   char *name, *val;
   for (name = strtok(NULL, "="); name != NULL; name = strtok(NULL, "=")) {
      val = strtok(NULL, "&");
      if (val != NULL) parameters.put(name, val);
   }
}

void isAuthorized(ObUserSession &user, ObResourceRequest &res, const char
     *authzParmString){
   ObBoolean_t authz = ObFalse;
   if (authzParmString != NULL) {
      ObMap authzParameters;
      char *authzParms = strdup(authzParmString);
      char *name, *val;
      for (name = strtok(authzParms, "="); name != NULL; name = strtok(NULL,
           "=")) {
         val = strtok(NULL, "&");
         if (val != NULL) authzParameters.put(name, val);
      }
      authz = user.isAuthorized(res, authzParameters);
   } else {
      authz = user.isAuthorized(res);
   }
   if (authz) {
      cout << "GRANTED\n";
   } else {
      cout << "DENIED\n";
      cout << "ERROR: " << user.getErrorMessage() << "\n";
      if (user.getError() == ObUser_ERR_NEED_MORE_DATA) {
         ObMap *pRequiredParameters = res.getAuthorizationParameters();
         cout << "EXPECTED PARAMETERS:";
         ObMapIterator iter(*pRequiredParameters);
         while (iter.hasMore()) {
            const char *name = NULL, *value = NULL;
            iter.next(&name, &value);
            cout << " " << name;
         }
         cout << "\n";
         delete pRequiredParameters;
      }
   }
   cout << "ACTIONS:";
   const char **actionTypes = user.getActionTypes();
   for (int i = 0; actionTypes[i] != NULL; i++) {
      const ObMap &actions = user.getActions(actionTypes[i]);
```

```
      ObMapIterator iter(actions);
      while (iter.hasMore()) {
         cout << "\n";
         const char *name = NULL, *value = NULL;
         iter.next(&name, &value);
         cout << actionTypes[i] << ": " << name << "=" << value;
      }
   }
}

void testLogin(const char *userid, const char *password, const char *method,
      const char *url, const char *location, const char *authzParmString) {
   char *urlCopy            = strdup(url);
   const char *resourceType = NULL;
   const char *resource     = NULL;
   ObMap parameters;
   parseUrl(urlCopy, &resourceType, &resource, parameters);
   ObResourceRequest res(resourceType, resource, method, parameters);
   if (res.isProtected()) {
      ObAuthenticationScheme authnScheme(res);
      if (authnScheme.isBasic()) {
         cout << "BASIC REALM : " << authnScheme.getChallengeParameter("realm")
            << "\n";
         ObMap credentials;
         credentials.put("userid", userid);
         credentials.put("password", password);
         ObUserSession user(res, credentials, location);
         if (user.getStatus() == ObUser_LOGGEDIN) {
            if (sessionToken != NULL) free((void *) sessionToken);
            sessionToken = strdup(user.getSessionToken());
            cout << "SESSION TOKEN : " << sessionToken << "\n";
            isAuthorized(user, res, authzParmString);
         } else {
            cout << "LOGIN FAILED: " << user.getErrorMessage() << "\n";
         }
      } else {
      cout << "RESOURCE SCHEME NOT BASIC";
      }
   } else {
      cout << "NOT PROTECTED";
   }
   free(urlCopy);
}

void testToken(const char *token, const char *method, const char *url) {
   if (sessionToken != NULL) free((void *) sessionToken);
   sessionToken = token;
   char *urlCopy = strdup(url);
   const char *resourceType = NULL;
   const char *resource     = NULL;
   ObMap parameters;
   parseUrl(urlCopy, &resourceType, &resource, parameters);
   ObResourceRequest res(resourceType, resource, method, parameters);
```

```
   if (res.isProtected()) {
      ObUserSession user(sessionToken);
      if (user.getStatus() == ObUser_LOGGEDIN) {
         cout << "USER: " << user.getUserIdentity() << "\n";
         if (user.getLocation() != NULL) {
            cout << "LOCATION: " << user.getLocation() << "\n";
         } else {
            cout << "LOCATION: (none)\n";
         }
         isAuthorized(user, res, NULL);
      } else {
         cout << "BAD TOKEN";
      }
   } else {
      cout << "NOT PROTECTED";
   }
   free(urlCopy);
}

void setLocation(const char *location) {
   ObUserSession user(sessionToken);
   user.setLocation(location);
   sessionToken = strdup(user.getSessionToken());
}

void showConfig() {
   cout << "CONFIGURATION:\n";
   cout << "The current version of SDK is " << ObConfig::getSDKVersion() << "\n";
   cout << "The current version of NAP is " << ObConfig::getNAPVersion() << "\n";
   ObMapIterator iter(ObConfig::getAllItems());
   while (iter.hasMore()) {
      const char *name, *val;
      iter.next(&name, &val);
      cout << name << ": " << (val != NULL ? val : "(none)") << "\n";
   }
}

void help() {
   cout << "EXPECT ONE OF\n";
   cout << "<userid> <password> <method> <url> [<location> [<authz-parameters>]]
        (sets sessionToken)\n";
   cout << "<sessionToken> <method> <url>      (sets sessionToken)\n";
   cout << "<method> <url>                     (uses prior sessionToken)\n";
   cout << "setLocation <newLocation>          (uses prior sessionToken)\n";
   cout << "showconfig\n";
   cout << "quit\n";
   cout << "exit\n";
}

int innerMain(int argc, char *argv[]) {
   float timeMilliSec;
#ifdef _WIN32
   struct _timeb startTime;
```

```
   struct _timeb stopTime;
   _ftime(&startTime);
#else
   struct timeval startTime;
   struct timeval stopTime;
   gettimeofday(&startTime, NULL);
#endif
   try {
      if (argc == 2) {
         if (strcmp(argv[1], "quit") == 0 || strcmp(argv[1], "exit") == 0) {
            return 1;
         } else
         if (strcmp(argv[1], "showconfig") == 0) showConfig();
      } else if (argc == 3) {
         if (sessionToken != NULL) {
            if (strcmp(argv[1], "setLocation") == 0) {
               setLocation(argv[2]);
            } else {
               testToken(strdup(sessionToken), argv[1], argv[2]);
            }
         } else {
            cout << "NO PRIOR LOGIN";
         }
      } else if (argc == 4) {
         testToken(argv[1], argv[2], argv[3]);
      } else if (argc == 5) {
         testLogin(argv[1], argv[2], argv[3], argv[4], NULL, NULL);
      } else if (argc == 6) {
         testLogin(argv[1], argv[2], argv[3], argv[4], argv[5], NULL);
      } else if (argc == 7) {
         testLogin(argv[1], argv[2], argv[3], argv[4], argv[5], argv[6]);
      } else {
         help();
      }
   }
   catch (ObAccessException *e) {
      cout << "EXCEPTION: " << e->toString();
      delete (e);
   }
#ifdef _WIN32
   _ftime(&stopTime);
   timeMilliSec = ((float) (stopTime.time    - startTime.time) * 1000) +
      (float) (stopTime.millitm - startTime.millitm);
#else
   gettimeofday(&stopTime, NULL);
   timeMilliSec = (((float) (stopTime.tv_sec - startTime.tv_sec) * 1000) +
      ((float) (stopTime.tv_usec - startTime.tv_usec) / 1000));
#endif
   cout << "\nTIME : " << timeMilliSec << " milliseconds";
   return 0;
}

int main(int argc, char *argv[]) {
```

```
   try {
      ObConfig::initialize();
      if (argc == 1) {
#        define MAX_ARGS 6
#        define MAX_INPUT_CHARS 1000
         int   ac;
         char *av[MAX_ARGS];
         char inputString[MAX_INPUT_CHARS];
         char *arg;
         help();
         int stop = 0;
         while (stop == 0) {
            cout << "\n>";
            cin.getline(inputString, MAX_INPUT_CHARS);
            av[0] = (char *) "access_test_cplus";
            ac = 1;
            for (arg = strtok(inputString, " ");
               arg != NULL && ac <= MAX_ARGS;
               arg = strtok(NULL, " ")) {
               av[ac] = arg;
               ac++;
            }
            if (ac > 1) {
               stop = innerMain(ac, av);
               cout << "\n";
            }
         }
      } else {
         innerMain(argc, argv);
      }
      cout << "\n";
      if (sessionToken != NULL) free((void *) sessionToken);
      ObConfig::shutdown();
   }
   catch (ObAccessException *e) {
      cout << "EXCEPTION: " << e->toString();
      delete (e);
   }
   return 0;
}
```

Annotated Code

If the host machine for the AccessGate is running the Win32 platform, then disable the warning relating to the deprecation of old i/o streams.

```
#ifdef _WIN32
#pragma warning(disable : 4995)
#endif
```

Include several standard C++ libraries. In addition to basic functionality, they cover input/output, strings, and streams.

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <iostream.h>
```

Import the time-handling library appropriate for the operating system platform (Win32 or other) used by the machine hosting the AccessGate.

```
#ifdef _WIN32
#   include <sys/timeb.h>
#else
#   include <time.h>
#endif
```

Include the C++ implementation of the Access Server API.

```
#include <obaccess_api_cplus.h>
```

Set to NULL the pointer to the current session token.

```
const char *sessionToken = NULL;
```

The “parseURL” method breaks the string “urlCopy” into its constituent elements, which are illustrated by the following:

```
[resourceType:][//host[:port]]/resource.
```

“urlCopy” might also contain an optional set of parameters in the following form:

```
[?p1=v1&p2=v2...].
```

where *p* represents the name of the parameter and *v* represents the value associated with that parameter name. In any case, “urlCopy” is optional.

```
void parseUrl(char *urlCopy, const char **resourceType, const char **resource,
     ObMap &parameters) {
```

Create pointers to the first colon (:) and double slash (//) within urlCopy and name them “colon” and “slashes,” respectively.

```
   char *colon   = strstr(urlCopy, ":");
   char *slashes = strstr(urlCopy, "//");
```

Determine whether urlCopy contains a colon but not a double slash, or whether it contains a colon which precedes a double slash.

```
   if (colon != NULL && (slashes == NULL || colon < slashes)) {
```

If either of the preceding statements is true, we can assume that urlCopy follows the form *resourceType*:[//host[:port]]/resource[?p1=v1&p2=v2...].

Therefore, we need to return a pointer to the token “resourceType,” which consists of everything in the string “urlCopy” prior to the delimiter colon (:).

```
      *resourceType = strtok(urlCopy, ":");
```

Each time the strtok method is invoked, it changes the delimiter to NULL. Therefore, we retrieve the second token in the string by specifying NULL as the start point and the question mark (?) as the closing delimiter. This returns the URL for the requested resource.

```
   *resource     = strtok(NULL,    "?");
```

If either part of the preceding "if" statement does not evaluate to true, we can assume that urlCopy follows the form [//host[:port]]/resource[?p1=v1&p2=v2...].

Therefore, we set resourceType to HTTP, and set a pointer to the token "resource," which consists of everything in "urlCopy" that precedes the delimiter question mark (?).

```
   } else {
      *resourceType = "http";
      *resource     = strtok(urlCopy, "?");
   }
```

Now parse the optional parameters section of the urlCopy string. Do this by assigning to "name" everything between NULL (formerly the question mark delimiter) and the equal sign (=), which is the new closing delimiter. As long as "name" is not empty, assign the next token to "val," this time using the ampersand (&) as the closing delimiter. Repeat this process until "name" is returned as NULL. This indicates that the end of the string has been reached.

```
   char *name, *val;
   for (name = strtok(NULL, "="); name != NULL; name = strtok(NULL, "=")) {
      val = strtok(NULL, "&");
      if (val != NULL) parameters.put(name, val);
   }
}
```

The isAuthorized method determines if the user submitting the request has permission to access a particular resource.

```
void isAuthorized(ObUserSession &user,ObResourceRequest &res, const char
     *authzParmString) {
```

Initiate the boolean variable "authz" as false.

```
ObBoolean_t authz = ObFalse;
```

Determine if the optional string "authzParmString" exists.

```
   if (authzParmString != NULL) {
```

If "authzParmString," which takes the format p1=v1&p2=v2...., does exist, break it into its constituent name:value pairs. Accomplish this by invoking the strtok method to read the name and value elements for each parameter into an ObMap structure named "authzParameters." Keep reading name:value pairs into "authzParameters" until the token "name" returns a NULL value, indicating that the end of the list has been reached.

```
   ObMap authzParameters;
   char *authzParms = strdup(authzParmString);
   char *name, *val;
   for (name = strtok(authzParms, "="); name != NULL; name = strtok(NULL,
        "=")) {
      val = strtok(NULL, "&");
      if (val != NULL) authzParameters.put(name, val);
   }
```

Determine if the user specified in the ObUserSession object “user” is authorized to access the ObResourceRequest object “res” by supplying the credentials specified in “authzParmString.”

```
   authz = user.isAuthorized(res, authzParameters);
}
```

If the optional “authzParmString” variable has not been supplied, determine if the user specified by the ObUserSession object “user” is authorized to access the resource requested through the ObResourceRequest structure “res.”

```
else {
   authz = user.isAuthorized(res);
}
```

If the user is authorized, report that access permission has been granted.

```
if (authz) {
   cout << "GRANTED\n";
```

Otherwise, report that access has been denied and append whatever error message has been returned.

```
} else {
   cout << "DENIED\n";
   cout << "ERROR: " << user.getErrorMessage() << "\n";
```

If ERR_NEED_MORE_DATA is returned, retrieve the required authorization parameters for the resource specified by “res.” Place the pointers for the name and value associated with each parameter into the ObMap structure pointed to by “pRequiredParameters.” Report on a new line the name (but not the corresponding value!) of each required parameter.

```
   if (user.getError() == ObUser_ERR_NEED_MORE_DATA) {
      ObMap *pRequiredParameters = res.getAuthorizationParameters();
      cout << "EXPECTED PARAMETERS:";
      ObMapIterator iter(*pRequiredParameters);
      while (iter.hasMore()) {
         const char *name = NULL, *value = NULL;
         iter.next(&name, &value);
         cout << " " << name;
      }
      cout << "\n";
```

To free the memory used by “pRequiredParameters.” which is no longer needed, invoke “delete.”

```
      delete pRequiredParameters;
    }
  }
```

Place the actions to be performed against the ObUserSession object "user" into the ObMap object "actions." Use ObMapIterator to report the name and value associated with each action.

```
  cout << "ACTIONS:";
  const char **actionTypes = user.getActionTypes();
  for (int i = 0; actionTypes[i] != NULL; i++) {
    const ObMap &actions = user.getActions(actionTypes[i]);
    ObMapIterator iter(actions);
    while (iter.hasMore()) {
      cout << "\n";
      const char *name = NULL, *value = NULL;
      iter.next(&name, &value);
      cout << actionTypes[i] << ": " << name << "=" << value;
    }
  }
}
```

The testLogin method demonstrates AccessGate login capabilities. Start by initializing or retrieving the necessary variables and constants.

```
void testLogin(const char *userid, const char *password, const char *method,
     const char *url, const char *location, const char *authzParmString) {
  char *urlCopy            = strdup(url);
  const char *resourceType = NULL;
  const char *resource     = NULL;
  ObMap parameters;
  parseUrl(urlCopy, &resourceType, &resource, parameters);
  ObResourceRequest res(resourceType, resource, method, parameters);
```

Determine whether the resource specified by the ObResourceRequest object "res" is protected.

```
  if (res.isProtected()) {
```

If the resource is protected, determine if the authentication scheme associated with "res" uses the basic challenge method.

```
    ObAuthenticationScheme authnScheme(res);
    if (authnScheme.isBasic()) {
```

If the authentication scheme does use the basic challenge method, report that fact along with the value of "realm," which is usually the name of the NetPoint authentication domain (an LDAP directory, for instance).

```
      cout << "BASIC REALM : " << authnScheme.getChallengeParameter("realm")
           << "\n";
```

Create an ObMap list named "credentials," and place into it the userid and password supplied for the current user session.

```
        ObMap credentials;
        credentials.put("userid", userid);
        credentials.put("password", password);
        ObUserSession user(res, credentials, location);
```

Determine whether the user is logged in, which is to say, the user has authenticated successfully.

```
        if (user.getStatus() == ObUser_LOGGEDIN) {
```

If the user is currently authenticated, determine whether the variable sessionToken exists. If it does, guard against stale data by deallocating the memory used by the sessionToken object.

```
           if (sessionToken != NULL) free((void *) sessionToken);
```

Retrieve a copy of the session token associated with “user” and assign it to the variable “sessionToken.”

```
           sessionToken = strdup(user.getSessionToken());
```

Report the contents of the serialized session token, which is an ASCII string representing the userid and password in encoded form.

```
           cout << "SESSION TOKEN : " << sessionToken << "\n";
```

Check whether the user is authorized to access the resource using the credentials passed through authzParmString.

```
           isAuthorized(user, res, authzParmString);
```

If the user has not authenticated successfully, report that along with the associated error message.

```
        } else {
           cout << "LOGIN FAILED: " << user.getErrorMessage() << "\n";
        }
```

If the challenge method is not basic, report that fact.

```
     } else {
     cout << "RESOURCE SCHEME NOT BASIC";
     }
```

If the resource is not protected, report that fact.

```
  } else {
     cout << "NOT PROTECTED";
  }
```

Deallocate the memory used by the variable “urlCopy.”

```
  free(urlCopy);
}
```

The testToken method demonstrates how authentication is achieved through an existing session token.

```
void testToken(const char *token, const char *method, const char *url) {
```

If the variable sessionToken exists, deallocate the memory assigned to it so as to avoid stale session data.

```
if (sessionToken != NULL) free((void *) sessionToken);
```

Initialize or retrieve the necessary variables and constants.

```
sessionToken = token;
char *urlCopy = strdup(url);
const char *resourceType = NULL;
const char *resource     = NULL;
ObMap parameters;
parseUrl(urlCopy, &resourceType, &resource, parameters);
ObResourceRequest res(resourceType, resource, method, parameters);
```

Determine if the resource specified by the ObResourceRequest structure "res" is protected.

```
if (res.isProtected()){
```

If the resource is protected, determine whether the user is logged in (which is to say, has successfully authenticated).

```
   ObUserSession user(sessionToken);
   if (user.getStatus() == ObUser_LOGGEDIN) {
```

If the user is authenticated, report the DN (Distinguished Name) of the user as it exists in the LDAP directory.

```
      cout << "USER: " << user.getUserIdentity() << "\n";
```

Determine if an IP address has been set for the user. Report the location, if that information it exists. Otherwise, report the location as "(none)".

```
      if (user.getLocation() != NULL) {
         cout << "LOCATION: " << user.getLocation() << "\n";
      } else {
         cout << "LOCATION: (none)\n";
      }
      isAuthorized(user, res, NULL);
```

If the user is not authenticated, report that the token is invalid.

```
   } else {
      cout << "BAD TOKEN";
   }
```

If the resource is not protected, report that fact.

```
} else {
   cout << "NOT PROTECTED";
}
```

Clean up by deallocating the memory used by the variable urlCopy.

```
   free(urlCopy);
}
```

Set the IP address of the user's machine.

```
void setLocation(const char *location) {
```

Create a session with the existing token.

```
   ObUserSession user(sessionToken);
```

Update the session token with the IP address returned by the "setLocation" method.

```
   user.setLocation(location);
   sessionToken = strdup(user.getSessionToken());
}
```

The showConfig method reports the configuration information currently set for the AccessGate. This includes the version of the Access Server SDK in use, the version of the NetPoint Access Control protocol in use, all of the other configuration parameters in the configuration file ObAccessClient.lst. This may include items such as the maximum number of resource request objects that can be cached, the maximum number of seconds before the user must re-authenticate, or the frequency with which the AccessGate checks to make sure its connection to the Access Server is still up.

```
void showConfig() {
   cout << "CONFIGURATION:\n";
   cout << "The current version of SDK is " << ObConfig::getSDKVersion() << "\n";
   cout << "The current version of NAP is " << ObConfig::getNAPVersion() << "\n";
   ObMapIterator iter(ObConfig::getAllItems());
   while (iter.hasMore()) {
      const char *name, *val;
      iter.next(&name, &val);
      cout << name << ": " << (val != NULL ? val : "(none)") << "\n";
   }
}
```

The help method reports the command-line syntax and options available for this application.

```
void help() {
   cout << "EXPECT ONE OF\n";
   cout << "<userid> <password> <method> <url> [<location> [<authz-parameters>]]
        (sets sessionToken)\n";
   cout << "<sessionToken> <method> <url>      (sets sessionToken)\n";
   cout << "<method> <url>                     (uses prior sessionToken)\n";
   cout << "setLocation <newLocation>          (uses prior sessionToken)\n";
   cout << "showconfig\n";
   cout << "quit\n";
   cout << "exit\n";
}
```

This is the "innerMain" method, which enables the program to execute with supplied command-line arguments or without command-line arguments in interactive mode. The "innerMain" method contains the code to execute a single operation; the main method reads in the opeations, then calls "innermain."

```
int innerMain(int argc, char *argv[]) {
```

Set up a variable to measure how long the application runs.

```
   float timeMilliSec;
```

Have the compiler set the time format correctly for the host machine platform running the AccessGate.

```
#ifdef _WIN32
   struct _timeb startTime;
   struct _timeb stopTime;
   _ftime(&startTime);
#else
   struct timeval startTime;
   struct timeval stopTime;
   gettimeofday(&startTime, NULL);
#endif
   try {
```

If precisely two strings exist in the array named "argc," determine whether the second string (arg[1]) is quit or exit. If the user has entered quit or exit, break out of this method.

```
      if (argc == 2) {
         if (strcmp(argv[1], "quit") == 0 || strcmp(argv[1], "exit") == 0) {
            return 1;
```

If the user has not entered quit or exit, determine whether the second string in the array "argv" is "showconfig." If so, invoke the showConfig method.

```
         } else if (strcmp(argv[1], "showconfig") == 0) showConfig();
```

If precisely three strings exist in "argc," we can assume that input follows the form <method> <URL>. Therefore, determine if a session token exists.

```
      } else if (argc == 3) {
         if (sessionToken != NULL) {
```

If a session token does exist, determine whether the user has entered "setLocation." If so, invoke the setLocation method.

```
            if (strcmp(argv[1], "setLocation") == 0) {
               setLocation(argv[2]);
```

If the user did not enter "setLocation" as the second element in the input string, invoke the testToken method, using the second and third arguments in "argc" as parameters.

```
            } else {
               testToken(strdup(sessionToken), argv[1], argv[2]);
            }
```

If a session token does not exist, report that the user is not currently authenticated.

```
         } else {
```

```
      cout << "NO PRIOR LOGIN";
   }
```

If precisely four strings exist in “argc,” we can assume that input follows the form <sessionToken> <method> <URL>. Therefore, invoke the testToken method, using the second through fourth strings in “argc” as parameters.

```
} else if (argc == 4) {
   testToken(argv[1], argv[2], argv[3]);
```

If precisely five strings exist in “argc,” we can assume that input follows the form <userid><password> <method> <URL>. Therefore, invoke the testLogin method, using the second through fifth strings in “argc” as parameters. Also, set the final two arguments for testLogin to NULL.

```
} else if (argc == 5) {
   testLogin(argv[1], argv[2], argv[3], argv[4], NULL, NULL);
```

If precisely six strings exist in “argc,” we can assume that input follows the form <userid><password> <method> <URL><location>. Therefore, invoke testLogin, using the second through sixth strings in “argc” as parameters. Set the final argument for testLogin to NULL.

```
} else if (argc == 6) {
   testLogin(argv[1], argv[2], argv[3], argv[4], argv[5], NULL);
```

If precisely seven strings exist in “argc,” we can assume that input follows the form <userid><password> <method> <URL><location><authzParameters>. Therefore, invoke testLogin, using the second through sixth strings in “argc” as parameters.

```
} else if (argc == 7) {
   testLogin(argv[1], argv[2], argv[3], argv[4], argv[5], argv[6]);
} else {
```

If the number of strings in “argc” is either one or more than seven, invoke the help method, which displays the command-line syntax for the application. This teaches the user how to enter the right type of information in the proper format.

```
   help();
}
```

If an error occurs in the main part of the innerMain method, report that fact, along with the error message “e.”

```
} catch (ObAccessException *e) {
   cout << "EXCEPTION: " << e->toString();
```

To free the memory used by “e,” which is no longer needed, invoke “delete.”

```
   delete (e);
}
```

Have the compiler set up the appropriate time-related functions for the platform in use.

```
#ifdef _WIN32
   _ftime(&stopTime);
   timeMilliSec = ((float) (stopTime.time    - startTime.time) * 1000) +
      (float) (stopTime.millitm - startTime.millitm);
#else
   gettimeofday(&stopTime, NULL);
   timeMilliSec = (((float) (stopTime.tv_sec - startTime.tv_sec) * 1000) +
      ((float) (stopTime.tv_usec - startTime.tv_usec) / 1000));
#endif
```

Report how long the application has been running.

```
   cout << "\nTIME : " << timeMilliSec << " milliseconds";
   return 0;
}
```

The main method ties everything together.

```
int main(int argc, char *argv[])
{
   try {
```

Initialize the Access Server SDK using the scope resolution operator to make sure we are calling the "initialize" method defined in Com.Oblix.Access.ObConfig.

```
      ObConfig::initialize();
```

Determine whether "argc" contains a single string.

```
      if (argc == 1) {
```

If it does, we are in interactive mode. Therefore, use the compiler to accept input lines containing no more than 1000 characters organized into a maximum of six (space-separated) arguments.

```
#        define MAX_ARGS 6
#        define MAX_INPUT_CHARS 1000
```

Initialize the appropriate variables. "ac" tracks the number of arguments in the input string. "inputString" stores the input retrieved from the command line, as long as it does not exceed 1,000 characters in length. "arg" stores each argument as it is parsed out of inputString. The array named "av" stores the space-separated arguments parsed from "inputString."

```
         int   ac;
         char *av[MAX_ARGS];
         char inputString[MAX_INPUT_CHARS];
         char *arg;
```

Invoke the help method so that user knows the required syntax and permissible range for command-line input.

```
         help();
         int stop = 0;
```

Retrieve the command-line input string named “inputString,” as long as it does not exceed the permitted maximum length. (The new line character that terminates input is not included in “inputString”).

```
    while (stop == 0) {
       cout << "\n>";
       cin.getline(inputString, MAX_INPUT_CHARS);
```

Assign a pointer to the string “access_test_plus” to the first element in the array named “av.” Casting is necessary here because the character string “access_test_cplus” is “const char *”, while the array “av” must be declared “char *”.

```
       av[0] = (char *) "access_test_cplus";
```

Using the space character ( ) as the delimiter, return the first argument in “inputString” and assign it to the second argument in “av” (av[1]). Repeat this sequence as long as substrings remain in “inputString,” and the maximum number of arguments permitted for “av” has not been exceeded.

```
       ac = 1;
       for (arg = strtok(inputString, " ");
           arg != NULL && ac <= MAX_ARGS;
           arg = strtok(NULL, " ")) {
          av[ac] = arg;
          ac++;
       }
```

When no more substrings exist to be extracted from “inputString,” determine whether any substring from “inputString” was assigned to the array “av.” If so, pass “av” and “ac” to the innermain method for processing, then store the return status in the variable “stop.”

```
       if (ac > 1) {
          stop = innerMain(ac, av);
          cout << "\n";
       }
    }
 }
```

Otherwise, invoke the innerMain method, passing it the current values for “argc” and “argv.” If a session token exists, deallocate the memory it uses, then shutdown the Access Server API.

```
 else {
    innerMain(argc, argv);
 }
 cout << "\n";
 if (sessionToken != NULL) free((void *) sessionToken);
 ObConfig::shutdown();
}
```

If an error occurs anywhere within the main method, report the appropriate error message.

```
   catch (ObAccessException *e) {
      cout << "EXCEPTION: " << e->toString();
```

To free the memory used by “e,” which is no longer needed, invoke “delete.”

```
      delete (e);
   }
```

End the application.

```
   return 0;
}
```

C++ Implementation Details

This section details the classes, constructors, methods, and parameters associated with the C++ implementation of the Access Server API. For an overview of the Access Server API classes, see “About the Access Server API” on page 268. The header file “obaccess_api_cplus.h,” also contains information on the C++ implementation of the API. This file resides at the following location:

SDK_install_dir\include

The C++ implementation of the Access Server API includes the classes listed in the following table:

Table 19 Overview of the Classes belonging to ObAccess (C++)

Class	Description
ObMap	Enables creation of and interaction with lists of name:value pairs.
ObMapIterator	Enables stepping through a list. You can determine the number of items in that list or retrieve a name:value pair from a specific position in that list.
ObAuthenticationScheme	Enables creation of and interaction with the structures that represent the authentication schemes used to authenticate users.
ObResourceRequest	Enables creation of and interaction with the structures that represent user requests to access resources.
ObUserSession	Enables creation of and interaction with structures representing sessions for users who have completed NetPoint authentication successfully.
ObConfig	Enables your application to initialize or shut down the Access Server. You can also obtain AccessGate configuration data from the Access Server.

Table 19 Overview of the Classes belonging to ObAccess (C++)

Class	Description
ObAccessException	Enables you to extract the entire error message string thrown by the Access Server API in response to an error. Alternatively, you can extract any or all of the embedded substrings (up to five) in an error message.

Note: For the C++ implementation of the Access Server API, you must invoke the "delete" method to "clean up" structures when they are no longer needed. See "About Memory Management" on page 268.

ObMap

ObMap facilitates the storage of Access Server API data by providing list structures into which name:value pairs can be written. The class also provides methods for retrieving information from the list, determining the number of items in that list, and copying the list. For a general discussion of ObMap, see page 270.

For a list of the messages thrown in response to errors by the C++ implementation of ObMap, see "C-Family Status and Error Message Strings" on page 397.

Constructors

The following table describes and details the constructors associated with the ObMap class.

Table 20 ObMap Constructors (C++)

Key Parameter	Details
NULL	**ObMap()** creates an empty list to hold name:value pairs. The name of this list is specified by the user when declaring the variable that represents this object.
otherMap	**ObMap(const ObMap &otherMap)** creates a copy of an existing list. The name of the new list is specified by the user. **Parameters**: **OtherMap**—The name of the list to be copied. **Returns**: A list to hold name:value pairs.

Methods

The following table lists the ObMap methods and associated parameters.

Table 21 ObMap Methods (C++)

Method	Details
get	**const char *get(const char *name) const** returns from the list the string value corresponding to the name specified. **Parameters**: **name**—The name member of the name:value pair for which the corresponding value is to be returned. **Returns**: The value member of the name:value pair. If the name is not found in the list, the method returns NULL.
put	**void put(const char *name, const char *val)** stores a name:value pair in the list. If the name is already in the list, its value is replaced; otherwise the name:value pair is added to the list. By implication a name can map to a just one value at any given time. **Parameters**: **name**—A string representing the name to be stored. **val**—A string representing the value to be stored.
size	**int size()const** returns the total number of the name:value pairs in the list.
copy	**ObMap *copy() const** creates a copy of ObMap. **Returns**: A pointer to the copy.

ObMapIterator

ObMapIterator enables you to step through a list created by ObMap and extract a name:value pair from a specific position in the list. Alternatively, you can use the ObMapIterator pointer to determine when the end of the list has been reached. For a general discussion of ObMapIterator, see page 271.

For a list of the messages thrown in response to errors by the C++ implementation of ObMapIterator, see “C-Family Status and Error Message Strings” on page 397.

Constructors

The following table lists details for the ObMapIterator constructors.

Table 22 ObMapIterator Constructors (C++)

Key Parameter	Details
map	**ObMapIterator(const ObMap &map)** creates an iterator (pointer) to a specified ObMap list. This pointer is used with ObMapIterator(next). **Parameters**: **map**—The name of the list.
other	**ObMapIterator(const ObMapIterator &other)** creates an iterator (pointer) to a copy of a specified ObMap list. This pointer is used with ObMapIterator(next). **Parameters**: **other**—The name of the list to be copied.

Methods

The following table lists the methods and associated details for ObMapIterator.

Table 23 ObMapIterator Methods (C++)

Method	Details
next	**void next(const char **name, const char **val)** returns the name:value pair from the current position in the list *and* moves the iterator to the following pair. **Returns**: Text strings for the next name:value pair.
hasMore	**ObBoolean_t hasMore() const** watches for the end of a list. It returns ObTrue if more name:value pairs exist in the map. It returns ObFalse if the end of the list has been reached.

ObAuthenticationScheme

ObAuthenticationScheme enables creation of and interaction with the structures that define the challenge methods and other parameters used by your AccessGate to authenticate users. For a general discussion of ObAuthenticationScheme, see page 272.

For a list of the messages thrown in response to errors by the C++ implementation of ObAuthenticationScheme, see “C-Family Status and Error Message Strings” on page 397.

Constructors

The following table lists details for the ObAuthenticationScheme constructors.

Table 24 ObAuthenticationScheme Constructors (C++)

Key Parameter	Details
resource	**public ObAuthenticationScheme(const ObResourceRequest &resource)** constructs an ObAuthenticationScheme object for the specified ObResourceRequest. **Parameters**: **resource**—The resource request for which the authentication scheme object is being constructed. **Returns**: A structure holding the object.
presource	**ObAuthenticationScheme(const ObResourceRequest *presource)** constructs an ObAuthenticationScheme object for the specified ObResourceRequest. **Parameters**: **resource**—The resource request for which the authentication scheme object is being constructed. **Returns**: A structure holding the object.
other	**ObAuthenticationScheme(const ObAuthenticationScheme &other)** is the copy constructor for this class. **Parameters**: **other**—The existing authentication scheme object to be copied. **Returns**: A structure holding a copy of the object.

Methods

The following table lists the ObAuthenticationScheme methods and associated details.

Table 25 ObAuthenticationScheme Methods (C++)

Methods	Details
getName	**const char *getName() const** returns the display name assigned to the authentication scheme during configuration.
getMask	**int getMask() const** returns a mask byte defining the security level.
requiresSecure Transport	**ObBoolean_t requiresSecureTransport() const** returns ObTrue if the scheme requires an SSL client connection; otherwise, it returns ObFalse. If the return flag is ObTrue, a redirectUrl is required.

Table 25 ObAuthenticationScheme Methods (C++)

Methods	Details
isBasic	**ObBoolean_t isBasic() const** returns a flag indicating if the authentication scheme requires only a userid-and-password challenge method. **Returns**: ObTrue if the scheme is Basic; otherwise, it returns ObFalse.
isCertificate	**ObBoolean_t isCertificate() const** returns ObTrue if the scheme requires a digital certificate; otherwise, it returns ObFalse.
isForm	**ObBoolean_t isForm() const** returns ObTrue if the authentication scheme uses customer-defined credential fields in an HTML login form (FORM-based authentication). Otherwise, it returns ObFalse.
isNone	**ObBoolean_t isNone() const** returns ObTrue if no credentials are required for authentication; otherwise, it returns ObFalse.
getLevel	**int getLevel() const** returns a numeric representation of the security level specified during authentication scheme configuration.
getRedirectUrl	**const char *getRedirectUrl() const** returns a string specified during configuration that represents the URL to which clients are redirected for authentication.
getChallenge Parameter	**const char *getChallengeParameter(const char *parameterName) const** returns the value for a challenge parameter that was specified by name during configuration. This parameter can be used to retrieve a space-separated list of context requests in the creds challenge parameter of an authorization scheme. The caller must parse the list to obtain the individual parameter names. **Parameters**: **parameterName**—The name of the challenge parameter. **Returns**: The corresponding challenge parameter value.
getAllChallenge Parameters	**const ObMap &getAllChallengeParameters() const** returns all the challenge parameters specified during configuration for a given authentication scheme. **Returns**: An ObMap list of name:value pairs.
getNumberOf Challenge Parameters	**int getNumberOfChallengeParameters() const** returns the total number of challenge parameters assigned to the Authentication Scheme during configuration.

ObResourceRequest

ObResourceRequest enables creation of and interaction with the structures that represent user requests to access resources. For a general discussion of ObResourceRequest, see page 275.

For a list of the messages thrown in response to errors by the C++ implementation of ObResourceRequest, see “C-Family Status and Error Message Strings” on page 397.

Constructors

The following table details the ObResourceRequest constructors.

Table 26 ObResourceRequest Constructors (C++)

Key Parameter	Details
operation	**ObResourceRequest(const char *resType,const char *resource,const char *operation)** constructs an ObResourceRequest object. The parameters are left as NULL values. **Parameters**: **resType**—The resource type (If resType is NULL, HTTP is used as a default.) **resource**—The name of the resource **operation**—The operation to be performed **Returns**: A structure holding the ObResourceRequest object.
parameters	**ObResourceRequest(const char *resType, const char *resource, const char *operation, const ObMap &parameters)** constructs an ObResourceRequest object. The parameter list is left as NULL values. **Parameters**: **resource**—The name of the resource. **operation**—The operation to be performed. **parameters**—A pointer to a list of context data (authorization parameters) **resType**—The resource type. (If resType is NULL, HTTP is used as a default.) **Returns**: A structure holding the object.
other	**ObResourceRequest(const ObResourceRequest &other)** is the copy constructor for this class. **Parameters**: **other**—The name of an existing ObResourceRequest structure to be copied. **Returns**: A structure holding a copy of the object.

Methods

The following table details the ObResourceRequest methods.

Table 27 ObResourceRequest Methods (C++)

Methods	Details
getResourceType	**const char *getResourceType() const** returns the resource type for the request.
getResource	**const char *getResource() const** returns the resource name for the request.
getOperation	**const char *getOperation() const** returns the name of the requested operation.
getParameters	**const ObMap &getParameters() const** returns a pointer to the first name:value pair in the list of parameters provided as a set of name:value pairs.
getNumberOf Parameters	**int getNumberOfParameters() const** returns a count of the number of pairs in the list.
isProtected	**ObBoolean_t isProtected()** returns ObTrue if the resource is protected by NetPoint policies; it returns ObFalse if it is not
getAuthorization Parameters	**ObMap *getAuthorizationParameters() const** When a response to the “ObUserSession.IsAuthorized” method includes a list of required context data, the list is cached in the ObResourceRequest object specified through the isAuthorized call. An AccessGate can obtain the list through the getAuthorizationParameters method. The AccessGate can add the appropriate values and pass the ObMap through a subsequent isAuthorized call. The caller is responsible for using the “delete” method to deallocate the ObMap object returned by getAuthorizationParameters. **Returns**: A list of name-value pairs with NULL values.
getNumberOf Authorization Parameters	**int getNumberOfAuthorizationParameters() const**; returns the number of required context data items.

ObUserSession

ObUserSession enables creation of and interaction with structures that represent sessions for users who have completed NetPoint authentication successfully. For a general discussion of ObUserSession, see page 277.

For a list of the messages thrown in response to errors by the C++ implementation of ObUserSession, see “C-Family Status and Error Message Strings” on page 397.

Constructors

The following table lists the ObUserSession constructors and associated details.

Table 28 ObUserSession Constructors (C++)

Key Parameter	Details
sessionToken	**ObUserSession(const char *sessionToken)** creates a user session object. **Parameters**: **sessionToken**—An ASCII text string that is parsed to obtain the result of authentication, which is to say, the user's DN and the level of the authentication scheme used. **Returns**: a structure holding a user session object. **Throws**: an *ObAccessException* if the user session object cannot be created for some reason or the sessionToken value is NULL.
resource	**ObUserSession(const ObResourceRequest &resource, const ObMap &credentials, const char *location = `NULL`)** creates a user session object, as described under "ObUserSession" on page 277. **Parameters**: **resource**—The resource object for which the user is being authenticated. **credentials**—The user credentials. **location**—The location of the user, if it needs to be specified. A valid DNS name or IP address can be used to specify the location of the user's machine. **Returns**: A structure holding a user session object. **Throws**: An ObAccessException if the user session object cannot be created for some reason or the resource object is NULL.
presource	**ObUserSession(const ObResourceRequest *presource, const ObMap &credentials, const char *location = `NULL`)** creates a user session object. **Parameters**: **presource**—The name of the resource **credentials**—The user credentials. **location**—The location of the user, if it needs to be specified. A valid DNS name or IP address can be used to specify the location of the user's machine. **Returns**: A user session object. **Throws**: An ObAccessException if the user session object cannot be created for some reason or the resource object is NULL.

Table 28 ObUserSession Constructors (C++)

Key Parameter	Details
&other	**ObUserSession(const ObUserSession &other)** creates a copy of a user session object. **Parameters**: **other**—The existing user session object to be copied. **Returns**: A copy of the user session object. **Throws**: An ObAccessException if the user session object cannot be created for some reason or the resource object is NULL.

Methods

The following table lists ObUserSession methods and associated details.

Table 29 ObUserSession Methods (C++)

Method	Details
getUserIdentity	**const char *getUserIdentity() const** returns the DN of the user's profile entry in the user directory.
getLocation	**const char *getLocation() const** returns the IP address of the user's web browser.
getAction	**const char *getAction(const char *actionType, const char *name) const** returns an action corresponding to the action name and type specified. **Parameters**: **actionType**—The type of action for which the value is returned. If this is left NULL, the default is headerVar. **name**—The name of the action for which the value is returned.
getActions	**const ObMap &getActions(const char *actionType)const** returns an ObMap list of action names and values in response to a specified action type. **Parameters**: **actionType**—The type of action for which the list is returned. If this is left NULL, the default is "headerVar".
getActionTypes	**const **getActionTypes() const** returns all the action types as an array of pointers to strings. The array is terminated by a NULL pointer.
getNumberofActions	**int getNumberOfActions(const char *actionType) const** returns the total number of actions of the specified action type that are associated with the user session. **Parameters**: **actionType**—The name of the action type for which a count of actions is needed. If this is left NULL, the default is headerVar.

Table 29 ObUserSession Methods (C++)

Method	Details
getLevel	**int getLevel() const** returns a numeric value representing the level of the authentication scheme used to authenticate the user.
getStartTime	**int getStartTime() const** returns the time at which the user was authenticated. This value is used to determine when a session expires. **Returns**: The number of seconds since midnight, January 1, 1970 since the user was authenticated.
getLastUseTime	**int getLastUseTime() const** returns the time since the most recent user request was authorized. This value is used to determine when an idle session expires. **Returns**: The number of seconds since midnight, January 1, 1970 since the most recent user request was authorized.
getStatus	**ObUserStatus_t getStatus() const** describes the current session status. **Returns**: An ObUserStatus_t value, such as logged out, logged in, login failed, or expired.
getError	**ObUserError_t getError() const** returns one of the ObUserError_t error values determined by the most recent authentication or authorization.
getErrorMessage	**const char *getErrorMessage(int err)** const returns the detailed error message for the authentication or authorization failure. The text of this message is derived by the Access API and is not for user modification. **Parameters**: **err**—The numerical error code corresponding to the authentication or authorization failure.
getSessionToken	**const char *getSessionToken() const** returns the saved, encrypted ASCII string representing the user session.
getUserIdentity	**const char *getUserIdentity() const** returns the Distinguished Name of the user's profile entry in the user directory.
setLocation	**void setLocation(const char *location)** sets the location of the user’s browser.

Table 29 ObUserSession Methods (C++)

Method	Details
isAuthorized (resource)	**ObBoolean_t isAuthorized() const ObResourceRequest &resource)** returns ObTrue if the user is authorized to perform an operation for a particular resource; otherwise, it returns ObFalse. **Parameters**: **resource**—The resource object whose authorization is to be checked. **Throws**: An ObAccessException, if the authorization check cannot be completed for any reason.
isAuthorized (presource)	**ObBoolean_t isAuthorized(const ObResourceRequest *presource)** returns ObTrue if the user is authorized to request an operation for a particular resource; otherwise, it returns ObFalse. **Parameters**: **presource**—The resource object whose authorization is to be checked. **Throws**: An ObAccessException, if the authorization check cannot be completed for any reason.
isAuthorized (...res, parameters)	**ObBoolean_t isAuthorized(const ObResourceRequest &res [const ObMap &parameters])** returns ObTrue if the user is authorized to request an operation for a particular resource; otherwise it returns ObFalse. **Parameters**: **res**—The resource object whose authorization is to be checked. **ObMap &parameters**—A list of name-value pairs that the Access Server API will send in the request context object to the Access Server for authorization. The parameter argument is optional. When specified, the name-value pairs in the parameters are passed to the Access Server for authorization. **Throws**: An ObAccessException if the authorization check cannot be completed for any reason.

Table 29 ObUserSession Methods (C++)

Method	Details
isAuthorized (...pRes, parameters)	**ObBoolean_t isAuthorized(const ObResourceRequest &pRes [const ObMap &parameters])** returns ObTrue if the user is authorized to request an operation for a particular resource; otherwise it returns ObFalse. **Parameters**: **pRes**—The resource object whose authorization is to be checked. **ObMap &parameters**—A list of name-value pairs that the Access Server API will send in the request context object to the Access Server for authorization. The parameter argument is optional. When specified, the name-value pairs in the parameters are passed to the Access Server for authorization. **Throws**: An ObAccessException if the authorization check cannot be completed for any reason.
logoff	**void logoff()** logs off the authenticated user and terminates the session.

ObConfig

ObConfig enables the application to initialize or shut down the Access Server or obtain AccessGate configuration data from the Access Server. For a list of AccessGate configuration parameters returned by ObConfig.getItem and ObConfig.getAllItems, see “Configuration Parameters” on page 281.

For a general discussion of ObConfig, see page 283.

No constructors exist for the C++ implementation of the class ObAccess.ObConfig.

For a list of the messages thrown in response to errors by the C++ implementation of ObConfig, see “C-Family Status and Error Message Strings” on page 397.

Methods

The following table details the methods associated with ObConfig.

Table 30 ObConfig Methods (C++)

Method	Details
initialize	**static void initialize(const char *installDir = NULL)** initializes the AccessGate, including reading all AccessGate configuration parameters into the ObConfig structure. See “Configuration Parameters” on page 281 , **Parameters**: **installDir**—A coded internal value provided for this parameter; otherwise, the value for the environment variable OBACCESS_INSTALL_DIR. **Throws**: An ObAccessException, if the user session object cannot be created, or if the OBACCESS_INSTALL_DIR is invalid.
shutdown	**static void shutdown()** disconnects the AccessGate from the Access Server, and releases memory and other resources.
getAllitems	**static ObMap &getAllItems()** reads all the configuration variables from the configuration file into an ObMap name:value list. See “Configuration Parameters” on page 281. **Throws**:An ObAccessException if an attempt is made to invoke the method before successful initialization takes place.
getSDKVersion	**static const char *getSDKVersion()** returns the SDK version as an internal value known to the API. **Throws**: An ObAccessException if an attempt is made to invoke the method before a successful initialization takes place.
getNAPVersion	**static const char *getNAPVersion()** returns, as an internal value known to the API, the version of the NetPoint Access Control protocol being used by the API. **Throws**: An ObAccessException if an attempt is made to invoke the method before a successful initialization takes place.
getNumberOf Items	**static int getNumberOfItems()** returns the total number of items that can be extracted from the configuration file. **Throws**: An ObAccessException if an attempt is made to invoke the method before a successful initialization takes place.

Table 30 ObConfig Methods (C++)

Method	Details
getItem	**static const char *getItem(const char* name)** returns a string representing the name of a configuration item listed in “Configuration Parameters” on page 281. **Parameters**: **name**—The name of a configuration item. **Throws**: An ObAccessException if an attempt is made to invoke the method before a successful initialization takes place.

ObAccessException

The ObAccessException class enables you to trap errors generated in connection with the Access Server API. The C++ implementation of ObAccessException allows you to return the full error message associated with an error code, or, in the case of the most recently generated error code, return up to five substrings from the full message so that you can embed them in custom error message text.

Constructors

The following table lists the constructors for the C++ implementation of the ObAccessException class.

Table 31 ObAccessException Constructors (C++)

Key Parameter	Details
code	**ObAccessException(ObAccessExceptionCode_t code, const char *p1 = NULL, const char *p2 = NULL, const char *p3 = NULL, const char *p4 = NULL, const char *p5 = NULL)** constructs an exception in response to an error code. **Parameters**: **code**—The number corresponding to the error that has occurred. **p1 to p5**—Parameters that can be inserted into the message string, if the string allows variable content. Parameters are intended for internal use only.
other	**ObAccessException(const ObAccessException &other)** copies an existing exception. **Parameters**: **other**—The name of an existing exception.

Methods

The following table lists the methods for the C++ implementation of ObAccessException.

Table 32 ObAccessException Methods (C++)

Parameter	Details
getCode	**ObAccessExceptionCode_t get code** returns the value of the last error code generated within the Access Server API. (This is also the code that appears in the exception created by the API).
getParameter	**const char *getParameter(int index)** returns the substring that normally appears in the full error message at the location "%index." You can insert such a substring into custom message strings you have created. **parameters**: **index**—The index number of the location where the substring normally appears in the full error message generated by the API.
toString	**const char *toString()** returns the full error message string (including all applicable substrings) for the most recent error code generated within the API. To allow reporting of the message, do not free the return value.
getCodeString	**const char *getCodeString(ObAccessExceptionCode_t code)** returns the error message corresponding to the specified ObAccessExceptionCode_t error code. Since you specify the error code, this doesn't have to be the most recent error message generated by the API. Since the string is returned verbatim for the ObAccessGate.msg file, in which al the substrings are set to NULL, it does not contain current values for the substrings p1 to p5. To allow reporting of the message, do not free the return value. **parameters**: **code**—The ObAccessExceptionCode_t error code corresponding to the error message to be returned from ObAccessGate.msg.

C Implementation Details

The functions belonging to the "pseudo-classes" in the C implementation of the Access Server API have been systematically named so as to parallel the naming scheme used by the paradigmatic C++ implementation of the Access Server API. For example, the C function "ObMap_get" corresponds to the C++ method "ObMap.get," and "ObUser_isAuthorized" in C corresponds to "ObUserSession.isAuthorized"in C++. In fact, the "class member functions" in the C implementation are merely opague pointers to methods in the C++ implementation.

The header file “obaccess_api_c.h” details the members of the “pseudo classes” belonging to the C implementation of the Access Server API. It can be found at the following location:

SDK_install_dir\include.

For a comparative discussion of the implementations of the Access Server API, see “About the Access Server API” on page 268.

Note: For the C-language implementation of the Access Server API, you must “clean up” structures that are no longer needed by invoking the appropriate “_free” function for structures when they are no longer needed. See “About Memory Management” on page 268.

ObMap_t

The ObMap_t “pseudo class” provides list structures to hold the various sets of name:value pairs used by the Access Server API. In addition to creating such structures, you can write to them, retrieve information from them, determine how many item pairs they contain, and copy their contents. Another function exists to deallocate the memory used by a list structure. To avoid memory leaks, use this destructor whenever you no longer need a list.

For a general discussion of the ObMap class, see page 270.

For a list of the messages thrown in response to errors by the C implementation of ObMap_t, see “C-Family Status and Error Message Strings” on page 397.

Functions

The following table details the functions associated with ObMap_t.

Table 33 ObAccess.ObMap_t Pseudo-Class Functions (C)

Function	Details
ObMap_new	**ObMap_t ObMap_new()** creates an empty list with a name specified by the user. (This function serves as the constructor for this pseudo class). **Returns**: A list to hold name:value pairs.
ObMap_get	**const char *ObMap_get(ObMap_t map, const char *name)** returns a string value in response to a specified name from a specified list. **Parameters**: **map**—A pointer to a list. **name**—The name half of the name:value pair in a specified list for which a corresponding value is to be returned.

Table 33 ObAccess.ObMap_t Pseudo-Class Functions (C)

Function	Details
ObMap_put	**void ObMap_put(ObMap_t map, const char *name, const char *val)** stores a name:value pair in a specified list. If the name already exists in the list, its value is replaced. **Parameters**: **map**—The name of the list. **name**—A string representing the name of the item to be stored. **val**—A string representing the value to be stored.
ObMap_size	**int ObMap_size(ObMap_t map)** returns a number of name:value pairs in a specified list. **Parameters**: **map**—The name of the list.
ObMap_copy	**ObMap_t ObMap_copy(ObMap_t map)** makes a copy of a specified list. **Parameters**: **map**—The name of the list. **Returns**: A pointer to the copy.
ObMap_free	**void ObMap_free(ObMap_t *pMap)** frees the memory occupied by a specified list. (This function serves as the destructor for this pseudo class). **Parameters**: **pMap**—Pointer to the list location.

ObMapIterator_t

The ObMapIterator_t “pseudo class” provides functions that enable you to place a pointer within a list structure, so as to count the number of items in the list. Other functions enable you to “step through” a list by pointing to successive items in the list, determine when the end of a list has been reached, and deallocate the memory used by the pointer.

For a general discussion of the ObMapIterator class, see “ObMapIterator” on page 271.

For a list of the messages thrown in response to errors by the C implementation of ObMapIterator_t, see “C-Family Status and Error Message Strings” on page 397.

Functions

The following table details the functions for ObMapIterator_t.

Table 34 ObAccess.ObMapIterator_t Pseudo-Class Functions (C)

Function	Details
ObMapIterator_new	**ObMapIterator_t ObMapIterator_new(ObMap_t map)** creates an iterator for a specified list and points it initially at the first name:value pair in the list. (This function serves as the constructor for this pseudo class). **Parameters**: **map**—The name of the list. **Returns:** A pointer to the list.
ObMapIterator_hasMore	**ObBoolean_t ObMapIterator_hasMore(ObMapIterator_t iter)** watches for the end of the list, returning ObTrue if name:value pairs exist beyond the current position of the iterator. It returns ObFalse when the iterator reaches the end of the list. **Parameters**: **iter**—A pointer to the next item in the list.
ObMapIterator_next	**void ObMapIterator_next(ObMapIterator_t iter, const char **name, const char **val)** returns a text string representing the name:value pair existing at the current iterator position in the list. This function then moves the iterator to the following pair. Technically speaking, the "const char **" parameters are pointers to variables that will be set to pointers to character strings. **Parameters**: **iter**—A pointer to the next item in the list. **name**—The address of the variable that will receive the pointer to the character string represented by "name." **val**—The address of the variable that will receive the pointer to the character string represented by "val."
ObMapIterator_free	**void ObMapIterator_free(ObMapIterator_t *pIter)** frees the memory used by the list. (This function serves as the destructor for this pseudo class). **Parameters**: **pIter**—A pointer to the list location.

ObAuthenticationScheme_t

The ObAuthenticationScheme "pseudo class" enables the creation of and interaction with the structures used to authenticate users. For a general discussion of ObAuthenticationScheme, see page 272.

For a list of the messages thrown in response to errors by the C implementation of ObAuthenticationScheme_t, see “C-Family Status and Error Message Strings” on page 397.

Functions

The following table details the functions belonging to ObAuthenticationScheme_t.

Table 35 ObAccess.ObAuthenticationScheme_t Functions (C)

Function	Details
ObAuthn_new	**ObAuthnScheme_t ObAuthn_new(ObResourceRequest_t resource)** constructs an ObAuthenticationScheme object that returns information on the specified ObResourceRequest, such as the challenge method required for authentication. **Parameters**: **resource**—The resource request for which the authentication scheme object is to be created.
ObAuthn_getName	**const char *ObAuthn_getName(ObAuthnScheme_t scheme)** returns the display name assigned to the authentication scheme during configuration. **Parameters**: **scheme**—A pointer to the specified authentication scheme.
ObAuthn_getMask	**int ObAuthn_getMask(ObAuthnScheme_t scheme)** returns the mask byte indicating the authorization challenge method and whether credentials must be sent over a secure connection. For details on the mask byte, see “ObAuthenticationScheme” on page 272. **Parameters**: **scheme**—A pointer to the specified authentication scheme.
ObAuthn_requiresSecureTransport	**ObBoolean_t ObAuthn_requiresSecureTransport (ObAuthnScheme_t scheme)** returns ObTrue if the specified authentication scheme requires credentials to be sent over a secure (SSL or TLS) connection; otherwise, it returns ObFalse. If a secure connection is required, a redirectUrl must be specified during authenticate scheme configuration. **Parameters**: **scheme**—A pointer to the specified authentication scheme.
ObAuthn_isBasic	**ObBoolean_t ObAuthn_isBasic(ObAuthnScheme_t scheme)** returns ObTrue if the challenge method for the specified authentication scheme is HTTP BASIC. (In other words, it requires only a userid and password as credentials). Otherwise, it returns ObFalse. **Parameters**: **scheme**—A pointer to he specified authentication scheme.

Table 35 ObAccess.ObAuthenticationScheme_t Functions (C)

Function	Details
ObAuthn_ isCertificate	**ObBoolean_t ObAuthn_isCertificate(ObAuthnScheme_t scheme)** returns ObTrue if the authentication scheme requires a digital security certificate; otherwise, it returns ObFalse. **Parameters**: **scheme**—A pointer to the specified authentication scheme.
ObAuthn_ isForm	**ObBoolean_t ObAuthn_isForm(ObAuthnScheme_t scheme)** returns ObTrue if the authentication scheme requires customer-defined credential fields in an HTML login form; otherwise, it returns ObFalse. **Parameters**: **scheme**—A pointer to the specified authentication scheme.
ObAuthn_ isNone	**ObBoolean_t ObAuthn_isNone(ObAuthnScheme_t scheme)** returns ObTrue if *no* credentials are required for authentication. If credentials *are* required, it returns ObFalse. **Parameters**: **scheme**—A pointer to the specified authentication scheme.
ObAuthn_ getLevel	**int ObAuthn_getLevel(ObAuthnScheme_t scheme)** returns a numeric representation of the authentication strength, as specified during authentication scheme configuration. **Parameters**: **scheme**—A pointer to the specified authentication scheme.
ObAuthn_ getRedirectUrl	**const char *ObAuthn_getRedirectUrl(ObAuthnScheme_t scheme)** returns a URL representing the location where secure authentication is to be performed. If secure authentication is not required by the specified authentication scheme, this value is set to NULL. **Parameters**: **scheme**—A pointer to the specified authentication scheme.
ObAuthn_ getChallenge Parameter	**const char *ObAuthn_getChallengeParameter (ObAuthnScheme_t scheme, const char *parameterName)** returns a value corresponding to a specified challenge parameter associated with a specified authorization scheme. **Parameters**: **scheme**—A pointer to the authentication scheme. **parameterName**—The name of the challenge parameter.
ObAuthn_ getAll Challenge Parameters	**ObMap_t ObAuthn_getAllChallengeParameters (ObAuthnScheme_t scheme)** returns a name:value list containing all the challenge parameters specified for the specified authentication scheme. **Parameters**: **scheme**—A pointer to the specified authentication scheme.

Table 35 ObAccess.ObAuthenticationScheme_t Functions (C)

Function	Details
ObAuthn_ getNumberOf Challenge Parameters	**int ObAuthn_getNumberOfChallengeParameters (ObAuthnScheme_t scheme)** returns the number of challenge parameters assigned to the specified authentication scheme. **Parameters**: **scheme**—A pointer to the specified authentication scheme.
ObAuthn_free	**void ObAuthn_free(ObAuthnScheme_t *pScheme)** frees the memory used by the specified authentication scheme, and sets the pointer value to NULL. **Parameters**: **pScheme**—A pointer to the specified authentication scheme.

ObResourceRequest_t

The ObResourceRequest_t "pseudo class" enables creation of and interaction with the structures that represent user requests to access resources. For a general discussion of ObResourceRequest, see page 275.

For a list of the messages thrown in response to errors by the C implementation of ObResourceRequest_t, see "C-Family Status and Error Message Strings" on page 397.

Functions

The following table details the functions associated with ObResourceRequest_t.

Table 36 ObAccess.ObResourceRequest_t Pseudo-Class Functions (C)

Function	Details
ObResource Request_new	**ObResourceRequest_t ObResourceRequest_new(const char *resType, const char *resource, const char *operation, ObMap_t parameters)** constructs an ObResourceRequest using the specified resource type, resource name, operation, and parameters. **Parameters**: **resType**—The resource type. (If resType is NULL, HTTP is used by default). **resource**—A pointer to the resource. **operation**—The operation to be performed against the resource. **parameters**—A pointer to a list of parameters associated with the resource request. **Returns:** Pointer to a structure holding the object.

Table 36 ObAccess.ObResourceRequest_t Pseudo-Class Functions (C)

Function	Details
ObResource_ getResource Type	**const char *ObResource_getResourceType (ObResourceRequest_t resource)** returns the resource type for the specified resource request. **Parameters**: **resource**—A pointer to the resource being requested.
ObResource_ getResource	**const char *ObResource_getResource (ObResourceRequest_t resource)** returns the name of the resource being requested through the specified resource request. **Parameters**: **resource**—A pointer to the resource being requested.
ObResource_ getOperation	**const char *ObResource_getOperation (ObResourceRequest_t resource)** returns the name of the operation to be invoked against the resource through the specified resource request. **Parameters**: **resource**—A pointer to the resource being requested.
ObResource_ getParameters	**const ObMap_t ObResource_getParameters (ObResourceRequest_t resource)** returns a pointer to the name:value list of parameters associated with the specified resource request. **Parameters**: **resource**—A pointer to the resource being requested.
ObResource_ getNumberOf Parameters	**int ObResource_getNumberOfParameters (ObResourceRequest_t resource)** returns the number items in the name:value parameter list associated with the specified resource request. **Parameters**: **resource**—A pointer to the resource being requested.
ObResource_ isProtected	**ObBoolean_t ObResource_isProtected (ObResourceRequest_t resource)** returns ObTrue if the resource is protected by NetPoint policies. Otherwise, it returns ObFalse. **Parameters**: **resource**—A pointer to the resource being requested.
ObResource_ free	**ObBoolean_t isCertificate() const ObResourceRequest_t *resource)** frees the memory used by the resource object and sets to NULL the pointer to the list of parameters associated with the specified resource request. **Parameters**: **resource**—A pointer to the resource being requested.

Table 36 ObAccess.ObResourceRequest_t Pseudo-Class Functions (C)

Function	Details
ObResource_ getAuthorization Parameters	**ObMap_t ObResource_getAuthorizationParameters (ObResourceRequest_t res)** returns a list of parameters for the particular authorization scheme associated with the specified resource request. All the parameter names are rturned with the associated values set to NULL. Once the returned value is no longer in use, you must use ObMap_free() to deallocate the ObMap_t object returned by getAuthorizationParameters(). **Returns:** A list of required credentials.
ObResource_ getNumberOf Parameters	**int ObResource_getNumberOfAuthorizationParameters (ObResourceRequest_t res)** returns the number of context parameters required for the specified resource request.
ObResource_free	**void ObResource_free(ObResourceRequest_t *pRes)** deallocates the memory for the specified ObResourceRequest structure.

ObUserSession_t

The ObUserSession_t *pseudo class* enables creation of and interaction with structures representing sessions for users who have successfully completed NetPoint authentication. For a general discussion of ObUserSession, see page 277.

For a list of the messages thrown in response to errors by the C implementation of ObUserSession_t, see “C-Family Status and Error Message Strings” on page 397.

Functions

The following table details the functions associated with ObUserSession:

Table 37 ObAccess.ObUserSession_t Pseudo-Class Functions (C)

Function	Details
ObUserSession_ authenticate	**ObUserSession_t ObUserSession_authenticate (ObResourceRequest_t resource, ObMap_t credentials, const char *location)** creates a user session object. **Parameters**: **resource**—The resource request object for which the user is being authenticated. **credentials**—User credentials. **location**—The location of the user, if it needs to be specified. A valid DNS name or IP address can be used to specify the location of the user's machine. **Returns:** A user session object. **Exception**: An internally generated *ObAccessException* if the user session object cannot be created for some reason or the resource object is NULL.
ObUserSession_ fromToken	**ObUserSession_t ObUserSession_fromToken(const char *sessionToken)** creates a user session object. **Parameters**: **sessionToken**—A session token, which is parsed to obtain the associated credentials and user location information. **Returns**: A user session object. **Exception:** An internally generated *ObAccessException* if the user session object cannot be created for some reason or the resource object is NULL.
ObUser_get UserIdentity	**const char *ObUser_getUserIdentity(ObUserSession_t user)** **Parameters**: **user**—A pointer to the user session object.
ObUser_get Location	**const char *ObUser_getLocation(ObUserSession_t user)** returns the IP address of the user's web browser. **Parameters**: **user**—A pointer to the user session object. **Returns:** IP address of the user's web browser.

Table 37 ObAccess.ObUserSession_t Pseudo-Class Functions (C)

Function	Details
ObUser_get Action	**const char *ObUser_getAction(ObUserSession_t user, const char *actionType, const char *name)** returns a value corresponding to the specified action name and action type for the specified user session. **Parameters**: **user**—A pointer to the user session object. **actionType**—The type of action for which the corresponding value is returned. (If this is left NULL, the default is headerVar). **name**—The name of the action for which the corresponding value is to be returned.
ObUser_get Actions	**const ObMap_t ObUser_getActions(ObUserSession_t user, const char *actionType)** returns a pointer to list of action name:value pairs corresponding to the specified action type for the specified user session. **Parameters**: **user**—A pointer to the user session object. **actionType**—The type of action for which the list is returned. If actionType is left NULL, the default is headerVar.
ObUser_get ActionTypes	**const **getActionTypes(ObUserSession_t user)** returns an array of pointers to strings. This array, which is terminated by a NULL pointer, represents all the action types associated with the specified user session. **Parameters**: **user**—A pointer to the user session object.
ObUser_get Numberof Actions	**int getNumberOfActions(ObUserSession_t user, const char *actionType)** returns the number of actions of the specified type that are associated with the specified user session. **Parameters**: **user**—A pointer to the user session object. **actionType**—The name of the action type for which a count of actions is needed. If this is left NULL, the default is headerVar.
ObUser_get Level	**int ObUser_getLevel(ObUserSession_t user)** returns a number representing the authentication level of the authentication scheme associated with the specified user session. **Parameters**: **user**—A pointer to the user session object.
ObUser_get StartTime	**int ObUser_getStartTime(ObUserSession_t user)** returns the number of seconds between midnight January 1, 1970 and the initial time the user was authenticated for the specified session. This value is used to determine session expiration. **Parameters**: **user**—A pointer to the user session object.

Table 37 ObAccess.ObUserSession_t Pseudo-Class Functions (C)

Function	Details
ObUser_get LastUseTime	**int ObUser_getLastUseTime(ObUserSession_t user)** returns the number of seconds between midnight January 1, 1970 and the most recent time the user was authenticated for the specified session. This value is used to determine expiration for an idle session. **Parameters**: **user**—A pointer to the user session object. **Returns:** A numeric value for the time.
ObUser_get Status	**ObUserStatus_t ObUser_getStatus(ObUserSession_t user)** returns one of the ObUserStatus_t values describing the status of the session, such as logged out, logged in, login failed, or expired. **Parameters**: **user**—A pointer to the user session object.
ObUser_get Error	**ObUserError_t ObUser_getError(ObUserSession_t user)** returns one of the ObUserError_t error values, as determined by the most recent authentication or authorization failure for the specified user session. **Parameters**: **user**—A pointer to the user session object.
ObUser_get ErrorMessage	**const char *ObUser_getErrorMessage(ObUserSession_t user)** returns a detailed error message associated with the most recent authentication or authorization failure for the specified user session. The text of this message is derived by the Access API and is not intended to be changed by the user. **Parameters**: **user**—A pointer to the user session object.
ObUser_is Authorized	**ObBoolean_t ObUser_isAuthorized(ObUserSession_t user, ObResourceRequest_t resource)** returns ObTrue if the user is authorized to request an operation for a particular resource; otherwise, it returns ObFalse. **Parameters**: **user**—A pointer to the user session object. **resource**—The name of the resource request object whose authorization is to be checked.

Table 37 ObAccess.ObUserSession_t Pseudo-Class Functions (C)

Function	Details
ObUser_is AuthorizedWith Parameters	**ObBoolean_t ObUser_isAuthorizedWithParameters (ObUserSession_t user, ObResourceRequest_t res, ObMap_t parameters)** returns ObTrue if the user is authorized to request an operation for a particular resource. Otherwise, it returns ObFalse. **Parameters**: **user**—A pointer to the user session object. **resource**—The name of the resource request object whose authorization is to be checked. **parameters**—Any data associated with the resource request. This parameter is optional.
ObUser_get SessionToken	**const char *ObUser_getSessionToken(ObUserSession_t user)** returns from a session token saved on the user's hard disk an ASCII string containing information on the specified user session. **Parameters**: **user**—The name of the user session object.
ObUser_set Location	**void ObUser_setLocation(ObUserSession_t user, const char *location)** sets the location of the user's browser. **Parameters**: **user**—A pointer to the user session object. **location**—A DNS or IP address representing the location of the user's browser.
ObUser_logoff	**void ObUser_logoff(ObUserSession_t user)**; logs off the authenticated user and terminates the specified user session. **Parameters**: **user**—A pointer to the user session object.
ObUser_free	**void ObUser_free(ObUserSession_t *puser)** frees the memory assigned for the list and sets the list pointer to NULL. **Parameters**: **puser**—A pointer to the user session object.

ObConfig_t

The ObConfig_t *pseudo class* enables your application to initialize or shut down the Access Server. You can also obtain AccessGate configuration data from the Access Server. See "Configuration Parameters" on page 281.

For a general discussion of ObConfig, see page 280.

For a list of the messages thrown in response to errors by the C implementation of ObConfig_t, see "C-Family Status and Error Message Strings" on page 397.

Functions

The following table lists the functions and details for the ObConfig pseudo class.

Table 38 ObAccess.ObConfig_t Pseudo-Class Functions (C)

Function	Details
ObConfig_ initialize	**void ObConfig_initialize(const char *installDir)** initializes the Access Server API. This includes reading all the configuration parameters listed in "Configuration Parameters" on page 281. **Parameters**: **installDir**—The root of the Access Server API installation. If no coded value is provided for this parameter, the value from the environment variable OBACCESS_INSTALL_DIR is used. **Throws**: An internally generated ObAccessException if the configuration object cannot be created for some reason, or if the OBACCESS_INSTALL_DIR is invalid.
ObConfig_ shutdown	**void ObConfig_shutdown()** disconnects the AccessGate from the Access Server and releases memory and other resources used by the AccessGate and the Access Server API.
ObConfig_ getAllitems	**ObMap_t ObConfig_getAllItems()** returns a pointer to the list of all the AccessGate configuration items listed in "Configuration Parameters" on page 281. **Throws**: An internally generated ObAccessException if an attempt is made to invoke the method before the Access Server SDK has been successfully initialized.
ObConfig_ getNumber OfItems	**int ObConfig_getNumberOfItems()** returns the number of items in the AccessGate configuration file. **Throws**: An internally generated ObAccessException if an attempt is made to invoke the method before the Access Server SDK has been successfully initialized.
ObConfig_ getItem	**const char *ObConfig_getItem(const char* name)** returns a value corresponding to a name specified from the list of AccessGate configuration items. **Parameters**: **name**—The name of the item whose value is to be extracted. **Throws**: An internally generated ObAccessException if an attempt is made to invoke the method before the Access Server SDK has been successfully initialized.
ObConfig_ getSDK Version	**const char *ObConfig_getSDKVersion()** returns the version of the Access Server SDK as an internal value known to the API.

ObAccessException_t

ObAccess exceptions occur when the Access API detects unexpected, unrecoverable problems such as not being able to connect to an Access Server. For a general discussion of ObAccessException, see “ObAccessException” on page 283.

C-language Error Handlers

For an AccessGate written using the C implementation of the Access Server API, you must write an ObAccessExceptionHandler_t function, which is called when an ObAccessException occurs. Otherwise, the C implementation of the API simply will not catch exceptions. Thus, if you use the C API to construct an object, and an exception occurs, that object will be returned empty, since no exception handler exists to report and otherwise handle the exception.

Note: If the calling program is written in C++, the calling program, rather than the AccessGate, which is written without any exception handler in C, *might* catch some exceptions.

The C version of ObAccessExceptionHandler_t in versions of NetPoint prior to version 6 has been deprecated, because it passed only the exception *code*, not the full exception. Consequently, “ObAccessException_getCodeString” could not insert any exception parameter data into the exception message.

NetPoint 6.x provides a new version of the exception handler for AccessGates using the C implementation of the Access Server API. This new version, ObAccessExceptionHandler2_t, passes the entire exception, so that ObAccessException_toString can display the exception message, complete with embedded parameters. When you create AccessGates, be sure to use ObAccessExceptionHandler2_t instead of the previous version.

The preferred way to write an exception handler is:

```
void myExceptionHandler(ObAccessException e){
   printf("EXCEPTION: %s\n", ObAccessException_toString(e));
   exit(1);
}
```

The following line then informs the API (in other words, it registers the callback function) as to the name of the exception:

```
ObAccessException_setHandler2(myExceptionHandler);
```

Functions

The following table lists the functions and details for the C implementation of the ObAccessException class.

Table 39 ObAccess.ObAccessException_t Pseudo-Class Functions (C)

Parameter	Details
ObAccess Exception Handler2_t	**typedef void (*ObAccessExceptionHandler2_t) (ObAccess ExceptionCode_t exception)** This is not actually a function. Rather, it defines a pointer to a C++ function used within the API. You implement this definition with your own code. See “C-language Error Handlers” on page 366. **Parameters**: None. Rather, the pointer name you specify is passed to the ObAccessException_setHandler2 function and used automatically by other functions when exceptions occur. The exception argument that appears in this function is the exception generated by the API.
ObAccess Exception_ setHandler2	**void ObAccessException_setHandler(ObAccessException Handler2_t handler)** connects exception handling to whatever activities the user has chosen to include in the user-written exception handler. **Parameters**: **handler**—A pointer to the user-written exception handler function.
ObAccess Exception_ getCode	**ObAccessExceptionCode_t ObAccessException_getCode (ObAccessException_t e)** returns the error code associated with the full exception generated by the API. **Parameters**: **e**—The exception provided by the API.
ObAccess Exception_ getParameter	**const char *ObAccessException_getParameter (ObAccessException_t e, int which)** returns a text substring corresponding to the specified exception and the index of the substring (1 to 5). **Parameters**: **e**—The exception provided by the API. **which**—The index of the parameter (1 to 5), for which a text string equivalent is needed.
ObAccess Exception_ toString	**const char *ObAccessException_toString(ObAccessException_t e)** returns the full error message string (including all applicable substrings) for the most recent error code generated within the API. To allow reporting of the message, do not free the return value. **Parameters**: **e**—The exception provided by the API.

Deprecated Functions

The following table lists the functions that existed in the C-language implementation of ObAccessException prior to NetPoint version 6. These functions have been deprecated and should not be used when you write new AccessGate code, because they do not support the extraction of indexed strings from error messages. They are listed here merely for developers seeking to understand compatibility issues for AccessGates prior to NetPoint version 6.

Table 40 Deprecated ObAccessException Functions (C)

Parameter	Details
ObAccessException Handler_t	This is not actually a function. Rather, it defines a pointer to a C++ function used within the API. In any case, do not use this function. It has been deprecated.
ObAccessException_ setHandler	Do not use this function. It has been deprecated.
ObAccessException_ getCodeString	Do not use this function. It has been deprecated.

C# Implementation Details

The following sections describe the C# (.NET) managed code implementation for the Access Server API.

For the most part, the classes in the C# version follow the pattern established by the Java implementation of the Access Server API. However, the C# version departs from the Java paradigm in the following significant ways:

- The enumerators used to specify various conditions are wrapped by managed classes
- The class ObMap is wrapped by ObDictionary
- The class ObMapIterator is wrapped by ObDictionaryEnumerator
- Certain method names have been changed to match the naming conventions used for .NET properties.

  Note: A .NET property resembles a Java member variable in that both enable the user to read and write values to an object, but a .NET property is implemented using the *get* and *set* methods.

- The *Mgd* suffix has been appended to all the managed classes. Thus, ObAuthenticationScheme. ObResourceRequest, ObUserSession, ObConfig,

and ObAccessException become, respectively, ObResourceRequestMgd, ObUserSessionMgd, ObConfigMgd, and ObAccessExceptionMgd.

- In contrast to the C and C++ development-language interfaces, but like Java, the C# environment features a garbage collection service which automatically cleans up objects when they are no longer needed. Therefore, you do not invoke the *delete* or *_free* methods to clean up unused structures. See "About Memory Management" on page 268.

The listings in this document can also be found in the header file obaccess_api_mgd.h, which resides in the following location:

```
SDK_install_dir/include
```

The classes common to both the Access Server API and Access Manager API are listed in the file obaccess_api_common_mgd.h, which is also in the directory: *SDK_install_dir*/include.

ObDictionary

ObDictionary provides hashtable into which key-and-value pairs (the .NET equivalent of the name:value pairs in Java hashtables) can be written. The class also provides methods for retrieving information from the dictionary hashtable, determining the number of items in that list, and copying the list.

The ObDictionary class is derived from the .NET IDictionary class and corresponds to the ObMap class in the Java and C++ implementations of the Access Server API. It also corresponds to the ObMap_t "pseudo class" in the C implementation of the Access Server API. For a general discussion of the ObMap class, see page 270.

For a list of the messages thrown in response to errors by the C# implementation of ObDictionary, see "C-Family Status and Error Message Strings" on page 397.

Constructors

The following table provides details for the ObDictionary constructor.

Table 41 ObDictionary Constructors (C#)

Key Parameter	Details
(none)	ObDictionary()
map	**ObDictionary(const ObMap &map)** using a name specified by the user, creates a copy of an existing list in the form of an ObDictionary object. **Parameters**: **map**—The name of the list to be copied.

Table 41 ObDictionary Constructors (C#)

Key Parameter	Details

Methods

The following table presents the ObDictionary methods and associated details

Table 42 ObDictionary Methods (C#)

Method	Details
get_Item	**__property virtual System::Object *get_Item(System::Object *key)** returns a string value that corresponds to a name supplied from a dictionary list. If the name is not found in the list, NULL is returned. The item is returned as the base class object. The user is responsible for casting this item to an appropriate form. **Parameters**: **key**—The key (or name) in a dictionary list for which a value is to be returned.
add	**virtual void Add (System::Object *key, System::Object *value)** stores a key-and-value pair in the list. If the name is already in the list, its value is replaced; otherwise the pair is added. **Parameters**: **key**—The name half of the item to be stored in the dictionary. **value**—The value half of the item to be stored.
get_Count	**__property int get_Count()** returns the total number of key-and-value pairs in the list.
Clone	**Object *Clone()** makes a copy of ObDictionary. **Returns**: A pointer to the copy.

ObDictionaryEnumerator

You use the ObDictionaryEnumerator class to locate the entries in a dictionary hashtable. You can also determine the number of items in that list or retrieve a key-and-value pair from a specific position in that list.

For the C++ managed classes version of the Access Server API, the ObMapIterator class is implemented as the ObDictionaryEnumerator class, which is derived from the IDictionaryEnumerator class in the .NET Framework class library. Instead of the name:value pairs found in a Java hashtable, an ObDictionaryEnumerator dictionary contains key-and-value pairs.

For a general discussion of the ObMapIterator class, see page 271.

For a list of the messages thrown in response to errors by the C# implementation of ObDictionaryEnumerator, see “C-Family Status and Error Message Strings” on page 397.

Constructors

The following table presents details for the ObDictionaryEnumerator constructor.

Table 43 ObDictionaryEnumerator Constructors (C#)

Key Parameter	Details
dictionary	**ObDictionaryEnumerator(ObDictionary *dict)** creates an enumerator, which initially points to the first item in a specified ObDictionary hashtable. **MoveNext** and other ObDictionaryEnumerator methods make use of this structure. **Parameters**: **dict**—The name of the ObDictionary hashtable you want to step through.

Methods

The following table lists the ObDictionaryEnumerator methods and associated details.

Table 44 ObDictionaryEnumerator Methods (C#)

Method	Details
MoveNext	**bool MoveNext()** watches for the end of the list. It returns ObTrue when additional key-and-value pairs exist in the dictionary list. It returns ObFalse when it reaches the final item in the list.
get_Current	**__property Object *get_Current()** returns an object that represents the dictionary entry currently referenced by the enumerator.
get_Entry	**__property DictionaryEntry get_Entry()** returns a dictionary key-and-value pair for the current dictionary entry.
get_Key	**__property Object *get_Key()** returns the key (name) value from the current dictionary entry. The user must then cast the returned object to the appropriate class.
get_Value	**__property Object *get_Value()** returns the value from the current dictionary entry. The user must then cast the returned object to the appropriate class.
Reset	**void Reset()** points the enumerator to the first entry in the dictionary list.

ObAuthenticationSchemeMgd

ObAuthenticationSchemeMgd structures enable users to store, pass, and retrieve information related to authentication schemes. An authentication scheme specifies how a user is challenged for a set of credentials. For a general discussion of the ObAuthenticationScheme class, see page 272.

For a list of the messages thrown in response to errors by the C# implementation of ObAuthenticationSchemeMgd, see “C-Family Status and Error Message Strings” on page 397.

Constructors

The following table provides details for the ObAuthenticationSchemeMgd constructor.

Table 45 ObAuthenticationSchemeMgd Constructors (C#)

Key Parameter	Details
pRes	**ObAuthenticationSchemeMgd(ObResourceRequestMgd *pRes)** creates an ObAuthenticationScheme object for the specified ObResourceRequest. **Parameters**: **pRes**—A pointer to the resource request for which the authentication scheme object is being constructed. **Returns:** A structure holding the object.

Methods

The following table lists the methods and associated details for the ObAuthenticationSchemeMgd class.

Table 46 ObAuthenticationSchemeMgd Methods (C#)

Method	Details
get_Name	**__property System::String *get_Name()** returns the display name assigned to the authentication scheme.
get_Mask	**__property int get_Mask()** returns the mask byte defining the security level of the authentication scheme.
get_Requires Secure Transport	**__property bool get_RequiresSecureTransport()** returns ObTrue if the authentication scheme requires an SSL client connection; otherwise, it returns ObFalse. When the return flag is ObTrue, a redirectUrl is required to implement Secure Transport.

Table 46 ObAuthenticationSchemeMgd Methods (C#)

Method	Details
get_IsBasic	**__property bool get_IsBasic()** returns ObTrue if the authentication scheme requires only a “HTTP basic” challenge method (in other words, it requires only a userid and password); otherwise it returns ObFalse.
get_IsCertificate	**__property bool get_IsCertificate()** returns ObTrue if the authentication scheme requires a digital security certificate; otherwise it returns ObFalse.
get_IsForm	**property bool get_IsForm()** returns ObTrue if the authentication scheme uses “HTML form” login (in other words, it uses customer-defined credential fields); otherwise it returns ObFalse.
get_IsNone	**__property bool get_IsNone()** returns ObTrue if *no* credentials are required for authentication. If credentials *are* required, it returns ObFalse.
get_Level	**__property int get_Level()** returns a number representing the level of authentication strength, as specified during authentication scheme configuration.
get_RedirectUrl	**__property System::String *get_RedirectUrl()** returns a string representing the URL to which clients are redirected for Secure Transport authentication.
getChallenge Parameter	**System::String *getChallengeParameter(System::String *parameterName)** returns the value for a parameter corresponding to the current challenge method. For instance, the creds parameter for the form challenge method retrieves a space-separated list of context-dependent login requests. You must parse this list to obtain the individual parameter names. **Parameters**: **parameterName**—The name of the parameter corresponding to the current challenge method.
get_AllChallenge Parameters	**__property ObDictionary *get_AllChallengeParameters()** returns an ObDictionary list containing a key-and-value pair for each of challenge parameter assigned to the authentication scheme.
get_NumberOf Challenge Parameters	**__property int get_NumberOfChallengeParameters()** returns the total number of challenge parameters assigned to the authentication scheme during configuration.
Clone	**Object* Clone()** crates a copy of the specified authentication scheme structure.

ObResourceRequestMgd

The constructors and methods for the ObResourceRequestMgd class enable creation of and interaction with the structures that represent user requests to access resources. For a general discussion of the ObResourceRequest class, see page 275.

For a list of the messages thrown in response to errors by the C# implementation of ObResourceRequestMgd, see “C-Family Status and Error Message Strings” on page 397.

Constructors

The following table lists the ObResourceRequestMgd constructors and associated details.

Table 47 ObResourceRequestMgd Constructors (C#)

Key Parameter	Details
op	**ObResourceRequestMgd(System::String *resType, System::String *res, System::String *op)** constructs an ObResourceRequest object. **Parameters**: **resType**—The resource type. (If resType is NULL, HTTP is used as a default.) **res**—The name of the resource. **op**—The operation to be performed. **Returns**: A structure representing the ObResourceRequest object.
parameters	**ObResourceRequestMgd(System::String *resType, System::String *res, System::String *op, ObDictionary *parameters)** constructs an ObResourceRequest object. **Parameters**: **resType**—The resource type. (If resType is NULL, HTTP is used as a default.) **res**—The name of the resource. **op**—The operation to be performed. **parameters**—A pointer to a list of parameters to be used. **Returns**: A structure representing the object.

Methods

The following table lists the methods and associated details belonging to ObResourceRequestMgd.

Table 48 ObResourceRequestMgd Methods (C#)

Method	Details
get_Resource Type	**__property System::String *get_ResourceType()** returns a string representing the resource type for the request.
get_Resource	**__property System::String *get_Resource()** returns the resource name for the request.
get_Operation	**__property System::String *get_Operation()** returns the name of the requested operation.
get_Parameters	**__property ObDictionary *get_Parameters()** returns a pointer to the first key-and-value pair in a list of parameters.
get_Number OfParameters	**__property int get_NumberOfParameters()** returns the number of pairs in the list.
get_Is Protected	**__property bool get_IsProtected()** returns ObTrue if the resource is protected by NetPoint policies; otherwise, it returns ObFalse. **Throws:** an ObAccessException in response to a fatal error such as failure to connect with the Access Server.
get_ Authorization Parameters	**__property ObDictionary *get_AuthorizationParameters()** when an IsAuthorized response includes a list of required context data, the list is cached in the ObResourceRequest object specified by the isAuthorized() call. An AccessGate can get the list through the get_AuthorizationParameters method. The AccessGate can add the appropriate values and pass the ObDictionary into a subsequent isAuthorized call. The caller is responsible for using delete to deallocate the ObDictionary object returned by get_AuthorizationParameters. **Returns:** List of key-and-value pairs with null values.
get_NumberOf Authorization Parameters	**__property int get_NumberOfAuthorizationParameters()** returns the number of required context data items.

ObUserSessionMgd

ObUserSession enables creation of and interaction with structures that represent sessions for users who have completed NetPoint authentication successfully. For a general discussion of ObUserSession, see page 277.

For a list of the messages thrown in response to errors by the C# implementation of ObUserSessionMgd, see “C-Family Status and Error Message Strings” on page 397.

Constructors

The following table presents ObUserSessionMgd constructor details.

Table 49 ObUserSessionMgd Constructors (C#)

Key Parameter	Details
sessionToken	**UserSessionMgd(System::String *sessionToken)** creates a user session object. **Parameters**: **sessionToken**—An ASCII text string that is parsed to obtain the credentials and location information. **Returns**: A structure holding a user session object. **Throws**: An *ObAccessException* if the user session object cannot be created for some reason, or if the sessionToken value is NULL.
credentials	**ObUserSessionMgd(ObResourceRequestMgd *pRes, ObDictionary *credentials)** creates a user session object. **Parameters**: **pRes**—The resource object requested by the user. **credentials**—User credentials. **Returns**: A structure holding a user session object. **Throws**: An ObAccessException if the user session object cannot be created for some reason, or if the resource object is NULL.
location	**ObUserSessionMgd(ObResourceRequestMgd *pRes, ObDictionary *credentials, System::String *location)** creates a user session object. **Parameters**: **pRes**—The name of the resource. **credentials**—User credentials. **location**—The location of the user, if it needs to be specified. A valid DNS name or IP address can be used to specify the location of the user's machine. **Returns**: A user session object. **Throws**: An ObAccessException if the user session object cannot be created for some reason, or if the resource object is NULL

Methods

The following table lists the methods and associated details for the ObUserSessionMgd class.

Table 50 ObUserSessionMgd Methods (C#)

Method	Details
get_UserIdentity	**__property System::String *get_UserIdentity()** returns the Distinguished Name of the user's profile entry in the user directory.
get_Location	**__property System::String *get_Location()** returns the location of the user. A valid DNS name or IP address can be used to specify the location of the user's machine.
getAction	**System::String *getAction(System::String *actionType, System::String *name)** returns an action corresponding to the name of the action and action type specified. **Parameters**: **actionType**—The type of action for which the value is returned. If this is left NULL, the default is headerVar. **name**—Name of the action for which the value is to be returned. **Returns:** A string representing the action.
getActions	**ObDictionary *getActions(System::String *actionType)**; returns an ObDictionary list of action names and values, given an action type. **Parameters**: **actionType**—The type of action for which the list is returned. If this is left NULL, the default is "headerVar".
get_ActionTypes	**__property System::String *get_ActionTypes()** returns an array of pointers to strings, representing all the action types. The array is terminated by a NULL pointer.
getNumber OfActions	**int getNumberOfActions(System::String *actionType)** returns the total number of actions belonging to the specified action type that are also associated with the current user session. **Parameters**: **actionType**—The name of the action type for which a count of actions is needed. If this is left NULL, the default is headerVar.
get_Level	**_property int get_Level()** returns a number representing the level of the current authentication scheme.
get_StartTime	**__property int get_StartTime()** returns the time when the user was authenticated, in seconds since midnight January 1, 1970. Used to determine a session expiration.

Table 50 ObUserSessionMgd Methods (C#)

Method	Details
get_LastUseTime	**__property int get_LastUseTime()** returns the time set when user request is authorized, in seconds since midnight January 1, 1970. Used to determine an idle session expiration.
get_Status	**__property ObUserStatusMgd *get_Status()** returns one of the ObUserStatus_t values describing the status of the session, such as logged out, logged in, login failed, or expired.
get_Error	**__property ObUserError_t get_Error()** returns one of the ObUserError_t error values determined by the most recent authentication or authorization.
get_ErrorMessage	**__property System::String *get_ErrorMessage()** returns a detailed error message pertaining to authentication or authorization failure. The text of this message is derived by the Access API and is not intended to be changed by the user.
IsAuthorized	**bool IsAuthorized(ObResourceRequestMgd *pRes)** determines if the user is authorized to perform an operation in reference to a particular resource. **Parameters**: **pRes**—The resource object whose authorization is to be checked. **Returns:** ObTrue if authorization succeeds, otherwise ObFalse. **Throws:** An ObAccessExceptionMgd if the authorization check cannot be completed for any reason.
IsAuthorizedWith Parameters	**bool IsAuthorizedWithParameters(ObResourceRequestMgd *pRes, ObDictionary *parameters)** determines if the user is authorized to request an operation for a particular resource. The parameters argument is optional. If specified, the key-and-value pairs in the parameters will be passed to the Access Server. **Parameters**: **pRes**—The resource object whose authorization is to be checked. **parameters**—A list of key-and-value pairs sent to the Access Server as part of the request-context object. **Returns:** ObTrue if authorization succeeds, otherwise ObFalse. **Throws:** An ObAccessException if the authorization check cannot be completed for any reason.
get_SessionToken	**__property System::String *get_SessionToken()** returns the saved encrypted ASCII representation of the user session. **Returns:** The ASCII string representing the user session.
LogOff	**void LogOff()** logs off the authenticated user and terminates the session.

ObConfigMgd

ObConfigMgd enables the application to initialize or shut down the Access Server or obtain AccessGate configuration data from the Access Server.

For a list of AccessGate configuration items see “Configuration Parameters” on page 281.

For a general discussion of the ObConfig class, see page 280.

For a list of the messages thrown in response to errors by the C# implementation of ObConfigMgd, see “C-Family Status and Error Message Strings” on page 397.

Constructors

There are no constructors for the C# implementation of this class.

Methods

The following table lists the methods and associated details for the ObConfigMgd class.

Table 51 ObConfigMgd Methods (C#)

Method	Details
initialize	**static void initialize(System::String *configDir)** initializes the AccessGate, including reading all parameters into the structure defined under “Configuration Parameters” on page 281. **Parameters**: **configDir**—If no coded value is provided for this parameter, the value from the environment variable OBACCESS_INSTALL_DIR is used. **Throws**: An ObAccessException if the user session object cannot be created for some reason, or if the OBACCESS_INSTALL_DIR is invalid.
shutdown	**static void shutdown()** disconnects the AccessGate from the Access Server, and releases memory and other resources.
getItem	**static System::String *getItem(System::String *name)** returns the value corresponding to the key from a key-and-value pair in an ObDictionary hashtable. For a list of possible items, see “Configuration Parameters” on page 281. **Parameters**: **name**—The name of a configuration item in a dictionary list. **Throws:** An ObAccessException if an attempt is made to invoke the method before a successful initialization is achieved.

Table 51 ObConfigMgd Methods (C#)

Method	Details
get_AllItems	**__property static ObDictionary *get_AllItems()** reads all the configuration variables from the configuration file into a named ObDictionary key-and-value dictionary list. See the list of possible items in “Configuration Parameters” on page 281. **Throws:** An ObAccessExceptionMgd exception, if an attempt is made to invoke the method before a successful initialization is achieved.
get_Number OfItems	**__property static int get_NumberOfItems()** returns the total number of items extracted from the configuration file. **Throws:** An ObAccessExceptionMgd exception, if an attempt is made to invoke the method before a successful initialization is achieved.
get_SDKVersion	**__property static System::String *get_SDKVersion()** returns the SDK version, an internal value known to the API. **Throws**: An ObAccessExceptionMgd exception, if an attempt is made to invoke the method before a successful initialization is achieved.
get_NAPVersion	**__property static System::String *get_NAPVersion()** returns a string representing the version of the NetPoint Access Control protocol being used by the Access Server API This internal value is known to the API. **Throws**: An ObAccessExceptionMgd exception, if an attempt is made to invoke the method before a successful initialization is achieved.

ObAccessExceptionMgd

This class enables you to extract the entire error message string thrown by the Access Server API in response to an error. Alternatively, you can extract from the full error message one or more (up to five) of the indexed substrings that may be embedded in the full error message.

For a general discussion of the ObAccessException class, see page 351.

Constructors

The following table lists the ObAccessExceptionMgd constructor and associated details for the class.

Table 52 ObAccessExceptionMgd Constructor (C#)

Parameter	Details
ex	**ObAccessExceptionMgd(ObAccessException *ex)** takes ownership of the passed-in value of ObAccessException and then cleans up the memory it uses. **Parameters**: **ex**—An ObAccessException.

Methods

The following table lists the details for the methods associated with ObAccessExceptionMgd.

Table 53 ObAccessExceptionMgd Methods (C#)

Method	Details
get_Code	**__property ObAccessExceptionCode_t get_Code()** returns the value of the last error code generated by the API.
getParameter	**System::String *getParameter(int index)** returns just the substring that would appear in a full error message at location %index. This allows you to obtain the substring in isolation, perhaps for insertion into a logged message along with customized text. **Parameters**: **Index**—The location where the substring (parameter) would ordinarily appear in the message generated by the API.
get_String	**__property System::String *get_String()** returns the error message string corresponding to the last error code generated by the API. This allows you to get the entire message as defined by the API, perhaps for insertion verbatim into an error log. Do not free the return value.
getCodeString	**System::String *getCodeString(ObAccessExceptionCode_t code)** returns the error message string corresponding to the specified error code, which need not be the last one generated within the API. Because the substring (parameter) pointers are all set to NULL, the ObAccessGate.msg file text is returned verbatim, without the substrings (parameters) inserted. Do not free the return value. **Parameters**: **code**—The number corresponding to the error message string that should be found. **Returns:** The verbatim ObAccessGate.msg file text for the specified error.

Java Implementation Details

The Java package that implements the Access Server API consists of the following:

- Three interfaces, which contain pure virtual methods and no implementation code
- Four implementing (base) classes, which inherit most of their member methods from the ObAccess interfaces
- One class that handles program errors

All of these classes implement the interface "java.lang.cloneable," and all except ObConfig and ObAccessException implement a corresponding Com.Oblix.Access interface.

For a general discussion of the classes in the Access Server API, see, "About the Access Server API" on page 268.

Note: Java Garbage Collection automatically deallocates the memory of Access Server API objects when they are no longer needed. For a discussion of how memory management is handled by the four implementations of the Access Server API, see "About Memory Management" on page 268.

Interfaces

In the Java programming language, an interface is a special class that contains methods, but does not contain the code to implement those methods. An interface does not allow the construction of objects, nor can you instantiate variables. Instead, one or more implementing classes inherit methods from the interface. Typically, each base class implements the methods it inherits from the interface in a fashion that distinguishes it from its sibling base classes.

At present, however, the three interfaces in the Access Server API each have just one implementing class. Therefore, polymorphism does not come into play for the Access Server API.

The name of each interface matches the name of its sole implementing class, except "interface" is appended at the end. Thus, the ObAuthenticationScheme base class implements ObAuthenticationSchemeInterface.

The following table correlates interfaces and corresponding base classes in the Access Server API.

Table 54 Java Interface Implementation for Com.Oblix.Access

Implementing ObAccess Base Class	Corresponding ObAccess Interface
ObAccessException	none

Table 54 Java Interface Implementation for Com.Oblix.Access

Implementing ObAccess Base Class	Corresponding ObAccess Interface
ObAuthenticationScheme	ObAuthenticationSchemeInterface
ObConfig	none
ObResourceRequest	ObResourceRequestInterface
ObUserSession	ObUserSessionInterface

ObAuthenticationSchemeInterface

ObAuthenticationSchemeInterface provides methods to enable the creation and manipulation of the structures used to authenticate users who have requested access to a resource. You cannot directly invoke any of the member methods in this interface. Instead, you invoke the corresponding member methods of ObAuthenticationScheme, the base class that implements ObAuthenticationSchemeInterface.

Methods

ObAuthenticationSchemeInterface contains the following methods: getName(), requiresSecureTransport(), isBasic(), isCertificate(), isForm(), isNone(), getLevel(), getRedirectUrl(), getNumberOfChallengeParameters(), getAllChallengeParameters(), and getChallengeParameter(). See "ObAuthenticationScheme" on page 385

ObResourceRequestInterface

ObResourceRequestInterface provides methods to enable manipulation of structures used to represent user requests for access to specified resources. You cannot directly invoke any of the member methods in this interface. Instead, you invoke the corresponding member methods of ObResourceRequest, the base class that implements ObResourceRequestInterface.

Methods

ObResourceRequestInterface contains the following methods: isProtected(), getResourceType(), getResource(), getOperation(), getParameters(), and getAuthorizationParameters(). See "ObResourceRequest" on page 387

ObUserSessionInterface

ObUserSessionInterface provides methods to enable the creation and manipulation of the structures that represent a user session. You cannot directly invoke any of the member methods in this interface. Instead, you invoke the corresponding member methods of ObUserSession, the base class that implements ObUserSessionInterface.

Methods

ObUserSessionInterface contains the following methods: getUserIdentity(), getLevel(), getLocation(), setLocation(), getStartTime(), getLastUseTime(), getNumberOfActions(), getActions(), getAction(), getActionTypes(), getStatus(), getError(), getErrorMessage(), getSessionToken(), and logoff(). See "ObUserSession" on page 390.

(java.util.Hashtable)

The Java implementation of the Access Server API does not include its own class to handle the storage and manipulation of API-related parameters, which is handled by ObMap and ObMapIterator in the C++ implementation of the API. The equivalent classes for the C implementations are ObMap_t and ObMapIterator, respectively. For the C# implementation, the equivalents are ObDictionary and ObDictionaryEnumerator, respectively.

By contrast, the Java implementation relies on the standard Java class java.util.Hastable to provide equivalent functionality in this area. For a discussion of ObMap and ObMapIterator, see "Implementations Compared" on page 268.

Constructors

The following table details the constructors in java.util.Hashtable that provide relevant hashtable functionality for Com.Oblix.Access. It lists only those constructors corresponding to constructors in the C-family implementations of the ObMap and ObMapIterator classes.

Table 55 java.util.Hashtable Constructors (Java)

Details
Hashtable() creates a new, empty hashtable with a default initial capacity of 11 items and load factor of 0.75. (The size of the hashtable is automatically increased when it is filled beyond 0.75 of its maximum capacity).

Methods

The following table details the methods in java.util.Hashtable that provide relevant hashtable functionality for Com.Oblix.Access. It lists only those methods corresponding to methods in the C-family implementations of the ObMap and ObMapIterator classes.

Table 56 java.util.Hashtable Methods (Java)

Method	Details
get	**get()** returns the value corresponding to the name of the specified item in the specified hashtable.
put	**put()** inserts a specified name:value pair the specified hashtable.
size	**size()** returns the number of items in the specified hashtable
(copy constructor)	**Hashtable(map_t)** copies a specified hashtable. **Parameter**: **map_t** - The name of the hashtable to be copied.

ObAuthenticationScheme

ObAuthenticationScheme enables you to create and manipulate structures that represent the authentication scheme associated with the resource requested by the specified user.

For a general discussion of ObAuthenticationScheme, see page 272.

For a list of the messages thrown in response to errors by the Java implementation of ObAuthentication Scheme, see “Java Status and Error Message Fields” on page 390.

Constructors

The following table presents the constructor for ObAuthenticationScheme.

Table 57 Com.Oblix.Access.ObAuthenticationScheme Constructors (Java)

Key Parameter	Details
res	**public ObAuthenticationScheme(ObResourceRequest res)** creates an ObAuthenticationScheme object for the specified ObResourceRequest. **Parameter**: **res** - The resource request object for which the authentication scheme object is being constructed. **Throws**: ObAccessException if the attempt to create the object fails or if resource object is NULL.

Methods

The following table details the methods belonging to the ObAuthenticationScheme class. The "clone" method implements a corresponding method inherited from the interface java.lang.Clonable. The method "setNativeHandle" is reserved for internal use only. All the other methods in ObAuthenticationScheme implement corresponding methods inherited from ObAuthenticationSchemeInterface or the ObAuthenticationScheme superclass java.lang.Object.

Table 58 Com.Oblix.Access.ObAuthenticationScheme Methods (Java)

Method	Details
clone	**public java.lang.Object clone()** clones ObAuthenticationScheme objects. **Throws**: java.lang.CloneNotSupportedException.
getAllChallenge Parameters	**public java.util.Hashtable getAllChallengeParameters()** returns a hashtable containing name:value pairs representing all the challenge parameters currently set for the specified authentication scheme.
getChallenge Parameter	**public java.lang.String getChallengeParameter(java.lang.String parameterName)** returns a value corresponding to the specified challenge parameter name for the specified authentication scheme. **Parameters**: **parameterName** - name of the challenge parameter
getLevel	**public int getLevel()** returns the authentication level currently set for the specified authentication scheme.
getName	**public java.lang.String getName()** returns the display name of the specified authentication scheme.

Table 58 Com.Oblix.Access.ObAuthenticationScheme Methods (Java)

Method	Details
getNumberOf Challenge Parameters	**public int getNumberOfChallengeParameters()** returns the number of challenge parameters currently set for the specified authentication scheme.
getRedirectUrl	**public java.lang.String getRedirectUrl()** returns the URL to which the user's browser is redirected for secure authentication.
isBasic	**public boolean isBasic()** returns true if the specified authorization scheme is Basic (in other words, it requires only userid and password for authentication); otherwise, it returns false.
isCertificate	**public boolean isCertificate()** returns true if the specified authorization scheme requires a digital certificate; otherwise, it returns false.
isForm	**public boolean isForm()** returns true if the specified authorization scheme requires HTTP FORM-based authentication; otherwise, it returns false.
isNone	**public boolean isNone()** returns true if the authentication scheme does not have a specified challenge method. If the resource *is* protected by an authorization scheme that does specify a challenge method, it returns false.
requiresSecure Transport	**public boolean requiresSecureTransport()** returns true if the authentication scheme requires a secure connection; otherwise, it returns false.
setNativeHandle	**public void setNativeHandle(int nativeHandle)** Users must not invoke this method; it is reserved of internal use only.
equals, getClass, hashCode, notify, notifyAll, toString, wait	All of the methods in the cell to the left were inherited by ObAuthenticationScheme from the superclass java.lang.Object.

ObResourceRequest

This class enables the creation, passing, and retrieval of structures that represent user requests to access resources. For a general discussion of ObResourceRequest, see "ObResourceRequest" on page 275.

For a list of the error messages thrown in response to errors by member methods of the Java implementation of ObResourceRequest, see "Java Status and Error Message Fields" on page 390.

Constructors

The following table lists the constructors for the ObResourceRequest class.

Table 59 Com.Oblix.Access.ObResourceRequest Constructors (Java)

Key Parameter	Constructor
operation	**public ObResourceRequest(java.lang.String resType, java.lang.String resName, java.lang.String operation)** constructs a ObResourceRequest object of the specified resource type, name and operation. **Parameters**: **resType** - The type of the requested resource, which can be HTTP, EBJ, or user-defined. When this value is NULL, the default value is HTTP. **res** - The name of the resource being requested. **operation** - The operation to be performed against the resource request object. This can be GET or POST for HTTP resources and EXECUTE for EJB resources. **Throws**: ObAccessException if an error occurs during native object creation or if either the resName or operation parameters are NULL.
parameters	**public ObResourceRequest(java.lang.String resType, java.lang.String resName, java.lang.String operation, java.util.Hashtable parameters)** constructs a ObResourceRequest object of the specified resource type, name, operation. When “parameters” is specified as an argument, a hashtable containing all the authorization parameters currently assigned to the resource is passed to the constructor. The Access Server uses this information to determine the applicable policies and authorization decisions. **Parameters**: **resType** - The type of the requested resource, which can be HTTP, EBJ, or user-defined. When this value is NULL, the default value is HTTP. **res** - The name of the resource being requested. **operation** - The operation to be performed against the resource request object. This can be GET or POST for HTTP resources and EXECUTE for EJB resources. **parameters** - a hashtable containing all the authorization parameters currently assigned to the resource. **Throws**: ObAccessException if an error occurs during native object creation or if either the resName or operation parameters are NULL.I

Methods

The following table lists the methods and details for the ObResourceRequest class. The clone method implements a corresponding method inherited from the interface Java.lang.Cloneable. The method getNativeHandle is reserved for internal use only. All the other methods in ObResourceRequest implement corresponding methods inherited from the interface ObResourceRequestInterface and the ObResourceRequest superclass java.lang.Object.

Table 60 Com.Oblix.Access.ObResourceRequest Methods (Java)

Method	Details
isProtected	**public boolean isProtected()** returns true if the resource specified by the ObResourceRequest structure is protected; otherwise it returns false.
getResourceType	**public java.lang.String getResourceType()** returns the resource type of the specified resource.
getResource	**public java.lang.String getResource()** returns the name of the resource associated with the specified resource request object.
getOperation	**public java.lang.String getOperation()** returns the name of the operation (such as GET or POST for HTTP) to be performed against the resource associated with the specified resource request object.
getParameters	**public java.util.Hashtable getParameters()** returns a hashtable containing name:value pairs representing all the parameters that have been set to define the resource associated with the specified resource request object.
getAuthorization Parameters	**public java.util.Hashtable getAuthorizationParameters()** returns a hashtable containing the parameters required for authorization of the resource associated with the specified resource request object. All the value halves of the name:value pairs representing these parameters are set to NULL. In other words, the hashtable contains only the names of the parameters.
getNumberOf Authorization Parameters	**public int getNumberOfAuthorizationParameters()** returns the number of parameters required for authorization of the resource associated with the specified resource request object. This total equals the number of items contained in the hashtable returned by the getAuthorizationParameters method.
clone	**public java.lang.Object clone()** clones ObResourceRequest objects. **Throws**: java.lang.CloneNotSupportedException.

Table 60 Com.Oblix.Access.ObResourceRequest Methods (Java)

Method	Details
equals, getClass, hashCode, notify, notifyAll, toString, wait	All of the methods in the cell to the left were inherited from the ObResourceRequest superclass **java.lang.Object**.

ObUserSession

ObUserSession enables creation of and interaction with structures representing sessions for users who have successfully completed NetPoint authentication. For a general discussion of ObUserSession, see page 277.

A list of error messages thrown by ObUserSession methods in response to errors follows immediately below.

Java Status and Error Message Fields

The following table lists the self-descriptive field names thrown in response to errors by the constructors and methods for the ObUserSession class. The syntax for declaring these fields is the following:

```
public static final int fieldname
```

where *fieldname* represents the specified status message string.

The following table lists the fields associated with the Java implementation of the ObUserSession class. The first five fields constants refer to session states, while the rest refer to errors

Table 61 Com.Oblix.Access.ObUserSession Fields (Java)

Field
AWAITINGLOGIN
LOGGEDIN
LOGGEDOUT
LOGINFAILED
EXPIRED
OK
ERR_UNKNOWN

Field
ERR_AUTHN_PLUGIN_DENIED
ERR_INSUFFICIENT_LEVEL
ERR_NOT_LOGGED_IN
ERR_SESSION_TIMEOUT
ERR_IDLE_TIMEOUT
ERR_DENY
ERR_PASSWORD_EXPIRED

Table 61 Com.Oblix.Access.ObUserSession Fields (Java)

Field	Field
ERR_NO_USER	ERR_PASSWORD_CHANGE_ON_RESET
ERR_USER_REVOKED	ERR_USER_LOCKED_OUT
ERR_WRONG_PASSWORD	ERR_NEED_MORE_DATA
ERR_INVALID_CERTIFICATE	ERR_INCONCLUSIVE

Constructors

The following table lists the ObUserSession constructors along with their associated parameters and details.

Table 62 Com.Oblix.Access.ObUserSession Constructors (Java)

Parameter	Details
NULL	**public ObUserSession()** is the default constructor for ObUserSession Object
sessionToken	**public ObUserSession(java.lang.String sessionToken)** used the specified session token to create an ObUserSession object. **Parameters**: **sessionToken** - A serialized representation of a user session **Throws**: ObAccessException if object creation fails or if session token is NULL.
location	**public ObUserSession(ObResourceRequest res, java.util.Hashtable Credentials, java.lang.String location)** constructs a user session object for the specified resource and credentials **Parameters**: **res** - The resource request object for which user is being authenticated. **credentials** - The user credentials formatted as name:value paris in a hashtable **location** - The location of the user, if it needs to be specified. A valid DNS name or IP address can be used to specify the location of the user's machine. **Throws**: ObAccessException if object creation fails or if session token is NULL.

Table 62 Com.Oblix.Access.ObUserSession Constructors (Java)

Parameter	Details
credentials	**public ObUserSession(ObResourceRequest res, java.util.Hashtable Credentials)** creates a user session object for the specified resource and credentials. **Parameters**: **res** - The resource object for which user is being authenticated. **credentials** - The name of the hashtable containing the name:value pairs that represent the user's credentials. **Throws**: ObAccessException if object creation fails or if session token is NULL.

Methods

The following table lists the methods and details associated with the Java implementation of the ObUserSession class.The clone method implements a corresponding method inherited from the interface java.lang.Clone. The method setNativeHandle is reserved for internal use only. All the other methods in ObUserSession implement corresponding methods inherited from ObUserSessionInterface or the ObUserSession superclass java.lang.Object.

Table 63 Com.Oblix.Access.ObUserSession Methods (Java)

Method	Details
clone	**public java.lang.Object clone()** clones ObUserSession objects. **Throws**: java.lang.CloneNotSupportedException.
getAction	**public java.lang.String getAction(java.lang.String actionType, java.lang.String name)** returns the name of an action that corresponds to the type of action that has been specified. **Parameters**: **actionType** - The action type corresponding to the action to be named. For instance, the action type can be "cookie" or "headerVar" for HTTP resources. If the type of the action is unspecified, the default type is "headerVar." **actionName** - The name of the action associated with the specified action type.
getActions	**public java.util.Hashtable getActions(java.lang.String actionType)** returns the names of all the actions of the specified action type that are currently set for the specified user session. **Parameters**: **actionType** -The type of action for which the actions set for the specified user session are returned. For instance, the action type can be cookie or headerVar for HTTP resources. If the type of the action is unspecified, the default type is "headerVar".

Table 63 Com.Oblix.Access.ObUserSession Methods (Java)

Method	Details
getAction Types	**public java.lang.String[] getActionTypes()** returns an array of action types for the specified user session.
getError	**public int getError()** returns the error number from the most recent authentication or authorization.
getError Message	**public java.lang.String getErrorMessage()** returns the detailed error message for the authentication or authorization failure.
getLast UseTime	**public int getLastUseTime()** returns the number of seconds between January 1, 1970 and the time when the user request was authorized. This value is used to determine when an idle session expires.
getLevel	**public int getLevel()** returns the level of the authentication scheme used to authenticate the user making the resource request.
getLocation	**public java.lang.String getLocation**() returns the location of the user. A valid DNS name or IP address can be used to specify the location of the user's machine.
getNumber OfActions	**public int getNumberOfActions(java.lang.String actionType)** returns the number of actions currently set for the specified action type. **Parameters**: **actionType** - The type of action. For instance, the action type can be cookie or headerVar for HTTP resources. If the type of the action is unspecified, the default type is "headerVar".
getSession Token	**public java.lang.String getSessionToken()** returns a serialized representation of user session.
getStart Time	**public int getStartTime()** returns the time when the user was authenticated; used to determine a session expiration.
getStatus	**public int getStatus()** returns the status of the specified session, such as logged out, logged in, login failed, or expired.
getUser Identity	**public java.lang.String getUserIdentity()** returns the Distinguished Name of the authenticated user's profile entry in the current LDAP directory.
isAuthorized	**public boolean isAuthorized(ObResourceRequest res)** returns true if the user is authorized to request the specified operation for the specified resource. Otherwise, returns false. **Parameters**: **res** - resource object being checked for authorization **Throws**: ObAccessException if the operation fails.

Table 63 Com.Oblix.Access.ObUserSession Methods (Java)

Method	Details
isAuthorized (parameters)	**public boolean isAuthorized(ObResourceRequest res, java.util.Hashtable parameters)** returns true if the user is authorized to request a specified operation against a specified resource when a set of additional parameters is specified. Otherwise, it returns false. **Parameters**: **res** - The resource object being checked for authorization **parameters** - A hashtable of parameter names and values **Throws**: ObAccessException if the operation fails.
logoff	**public void logoff()** logs off the authenticated user and terminates the session
setLocation	**public void setLocation(java.lang.String location)** sets the location of the user. A valid DNS name or IP address can be used to specify the location of the user's machine.
setNative Handle	**public void setNativeHandle(int nativeHandle)** Do not invoke this method. It is reserved for internal use only.
equals, getClass, hashCode, notify, notifyAll, toString, wait	All of the methods listed in the cell to the left have been inherited from the ObUserSession superclass **java.lang.Object**.

ObConfig

The ObConfig class allows you to store, pass, retrieve, and modify configuration information for your AccessGate.

Constructors

The following table presents the constructors for the ObConfig class.:.

Table 64 Com.Oblix.Access.ObConfig Constructors (Java)

public ObConfig() is the default constructor for the ObConfig Object.

Methods

The following table presents the methods for the ObConfig class.:

Table 65 Com.Oblix.Access.ObConfig Methods (Java)

Method	Details
getAllItems	**public static java.util.Hashtable getAllItems()** a hashtable containing name:value pairs representing all the configuration parameters currently in the AccessGate configuration file. **Throws**: ObAccessException if this method is invoked before initialization of the Access Server API succeeds**.**
getItem	**public static java.lang.String getItem(java.lang.String itemName)** returns a value corresponding to the specified configuration parameter. **Parameters**: **itemName** - the name of the configuration parameter whose value is being requested. **Throws**: ObAccessException if this method is invoked before initialization of the Access Server API succeeds**.**
getNAP Version	**public static java.lang.String getNAPVersion()** returns the version of the NetPoint Access Protocol that is in use.
getNumber OfItems	**public static int getNumberOfItems()** returns the number of configuration items currently set for the AccessGate. **Throws**: ObAccessException if this method is invoked before initialization of the Access Server API succeeds.
getSDK Version	**public static java.lang.String getSDKVersion()** returns version of the Access Server SDK that is in use.
initialize (installDir)	**public static void initialize(java.lang.String installDir)** initializes the Accessgate using the Access Server SDK whose root is *installDir*. (Sometimes it is helpful to specify the location of the SDK instance supporting your AccessGate, especially when multiple server instances, each one protected by a different AccessGate or WebGate, have been installed on the same host machine. **Parameters**: **installDir** - The directory on the AccessGate host machine where the AccessServerSDK is installed. **Throws**: ObAccessException if initialization fails**.**
initialize	**public static void initialize()** initializes the AccessGate using the environment variable OBACCESS_INSTALL_DIR for AccessGates running on either UNIX or WIndows host machines. **Throws**: ObAccessException if initialization fails.
shutdown	**public static void shutdown()** disconnects the AccessGate from the Access Server.

ObAccessException

The Java implementation of ObAccessException differs from the corresponding C-family implementations, in that the Java environment only allows the extraction of entire error message strings, and not the extraction of the individual, indexed substrings that can be extracted through the C-family implementations. For a general discussion of ObAccessException, see page 283.

An ObAccessException is thrown by the Access Server API whenever unexpected, unrecoverable errors occur between an AccessGate and an Access Server.

Constructors

The following table presents the constructor for the ObAccessException class.

Table 66 Com.Oblix.Access.ObAccessException Constructors (Java)

Key Parameter	Details
NULL	**public ObAccessException()** constructs an ObAccessException.
message	**public ObAccessException(java.lang.String message)** constructs an ObAccessException that includes a specified message. **Parameters**: **message** - The exception message string.

Inherited Methods

The Java implementation of ObAccessException has no native methods of its own. The following table presents the ObAccessException classes inherited from java.lang.Throwable and java.lang.Object.

Table 67 Com.Oblix.Access.ObAccessException Inherited Methods (Java)

Inherited Methods	Source
fillInStackTrace, getLocalizedMessage, getMessage, printStackTrace, toString	All of the methods in the cell to the left were inherited from the ObAccessException superclass java.lang.Throwable
equals, getClass, hashCode, notify, notifyAll, wait	All of the methods in the cell to the left were inherited from the ObAccessException superclass java.lang.Object

C-Family Status and Error Message Strings

This section lists constants that are common to all of the C-family implementations of the Access Server API (C, C++, and C#). The list presented here is also included in obaccess_api_defs.h, which resides in the following directory:

SDK_install_dir/include

Note: In Java, errors are handled by reference, rather than by code number. For example, ObAccessException_NOT_INITIALIZED, error 205, is referenced as ObAccessException.NOT_INITIALIZE. See “Fields” on page .

The exception codes in the following table are returned by the Access Server if it cannot perform an operation because of missing data or system problems.

Note: All of the strings the following table must be prepended with the string ObAccessException_. Thus, OK = 0 becomes ObAccessException_OK = 0.

Table 68 Access Server Error Codes

Exception Code	Exception Code
OK = 0	MISSING_RESOURCE
UNKNOWN = 200	MISSING_OPERATION
BAD_SESSION_TOKEN	BAD_LOCATION
NO_SCHEME_ID	NO_CLIENT_ID
NEED_PARAMETERS	JNI_ERROR
NOT_INITIALIZED	OUT_OF_MEMORY
CACHE_PROBLEM	MISSING_ITEM
NO_CONFIG_FILE	NO_MSG_CAT
NO_INSTALL_DIR_ENV	CLIENT_NOT_IN_DIR
NOT_PROTECTED	OBERROR

The exception codes in the following table are returned by the Access Server if it cannot perform an operation because of missing data or system problems.

Note: All of the strings the following table must be prepended with the string "ObAccessException_." Thus, "AS_UNKNOWN = 300" becomes "ObAccessException_AS_UNKNOWN = 30"

Table 69 AccessGate Error Codes

Exception Code	Exception Code
AS_UNKNOWN = 300	INVALID_SCHEME_PARAMETERS
ENGINE_DOWN	NO_USER
NOCODE	NONUNIQUE_USER
NULL_RESOURCE	USER_REVOKED
HOSTPORT_LOOKUP_FAILED	MISSING_OBCRED_PASSWORD
URL_LOOKUP_FAILED	WRONG_PASSWORD
SD_LOOKUP_FAILED	MISSING_PASSWORD
WROR_LOOKUP_FAILED	MISSING_CERTIFICATE,
WROR_AUTHENT_LOOKUP_FAILED	INVALID_CERTIFICATE
NO_AUTHENT_SCHEME	INVALID_SELECTION_FILTER
EXCEPTION	MISSING_AUTHN_PLUGIN
INVALID_SCHEME_ID	AUTHN_PLUGIN_ABORT
INVALID_SCHEME_MAPPING	AUTHN_PLUGIN_DENIED
AS_UNKNOWN = 300	AUTHN_PLUGIN_NO_USER
ENGINE_DOWN	

The codes in the following table are returned to indicate the status of a user session.

Note: All of the strings listed in the following table must be prepended with the string "ObUser_." Thus, "AWAITINGLOGIN = 0" becomes "ObUser_ AWAITINGLOGIN = 0".

Table 70 Session Status Codes

Status Code	Status Code
AWAITINGLOGIN = 0	LOGINFAILED
LOGGEDIN	EXPIRED

Table 70 Session Status Codes

Status Code	Status Code
LOGGEDOUT	

The following table lists the codes that describe problems that might occur in the authentication or authorization process.

Note: All of the strings listed in the following table must be prepended with the string ObUser_. Thus, OK = 0 becomes ObUser_OK = 0.

Table 71 Authentication and Authorization Error Codes

Error Code	Error Code
OK = 0	ERR_AUTHN_PLUGIN_DENIED
ERR_UNKNOWN = 100	ERR_INSUFFICIENT_LEVEL
ERR_NO_USER,	ERR_NOT_LOGGED_IN
ERR_USER_REVOKED	ERR_SESSION_TIMEOUT
ERR_WRONG_PASSWORD	ERR_IDLE_TIMEOUT
ERR_INVALID_CERTIFICATE	ERR_DENY

The following table lists the codes used to ensure consistent meaning of true and false within the Access Server API.

Table 72 ObBoolean Return Codes

Code	Code
ObFalse = 0	ObTrue = 1

Best Practices

This section presents a number of ways to avoid problems and to resolve the most common problems that crop up during development.

Avoiding Problems

Here are some suggestions for avoiding problems with the AccessGates you create:

- Make sure that your AccessGate attempts to connect to the correct Access Server.

- Make sure the configuration information on your Access Server matches the configuration information on your AccessGate. You can check the AccessGate configuration information on your Access Server, which is stored in the Oblix configuration directory, by navigating to NetPoint System Console > Access System Configuration > AccessGate Configuration, then clicking the name of the AccessGate whose configuration information you want to check. For details about AccessGate and Access Server configuration, see the *NetPoint 7.0 Administration Guide Volume 2*.
- To ensure clean connect and disconnect from the Access Server, use the "initialize" and "shutdown" methods in the ObConfig class.
- The environment variable, OBACCESS_INSTALL_DIR, *must* be set on your Windows or Solaris host machine so that you can compile and link your AccessGate. In general, you also want the variable to be set whenever your AccessGate is running.
- Use the exception handling features (try, throw, and catch) of the language used to write your custom AccessGate code to trap and report problems during development.

Thread Safe Code

Your AccessGate represents just one thread in your entire, multithreaded NetPoint application.

To ensure safe operation within such an environment, Oblix recommends that developers observe the following practices:

- Use a thread safe function instead of its single thread counterpart. For instance, use localtime_r instead of localtime.
- Specify the appropriate build environment and compiler flags to support multithreading. For instance, use -D_REENTRANT. Also, use -mt for Solaris platforms and /MD for Windows platforms.
- Take care to use in thread-safe fashion shared local variables such as FILE pointers.

Identifying and Resolving Problems

Here are some things to look at if your AccessGate fails to perform:

- Make sure that your Access Server is running. On Windows systems, you can check this by navigating to

  Computer Management > Services > *AccessServer*

  where *AccessServer* is the name of the Access Server to which you want to connect your AccessGate

- Check that the NetPoint domain policies your code assumes are in place and enabled for your Access System.
- Make sure you have read the Release Note that accompanies the Access System product you are working with. You may be running into a bug. If it is a bug Oblix already knows about, a workaround may appear in the release note.
- Check that your AccessGate is not being answered by a lower-level Access System policy which overrides the one you think you are testing.
- Check that you have run the configureAccessGate tool and that you did not make any input errors when you did so. See “Running the configureAccessGate Utility” on page 262 for a description of how to run the tool.
- The Access Tester in the Access Manager enables you to check which policy applies to a particular resource. For details about using the Access Tester and protecting resources with policy domains, see the *NetPoint 7.0 Administration Guide Volume 2*.

6 Access Management API

NetPoint provides programmatic access to most of the functions provided by the Access Manager GUI. You can use the Access Management API to create and manage policy domains and their contents. For example, you can write applications that use the programmatic interface instead of the GUI to create, modify, delete, and retrieve policy domains and their contents.

This chapter explains how to write applications that use the Access Management API. It contains these sections:

- "About the Access Management API" on page 404 describes the API and explains how it works.
- "Development Environment" on page 407 identifies the source and header files for the API and describes the build process.
- "Coding With the Access Management API" on page 410 describes some of the methods common to most classes which are included in the API. This section discusses the ObAccessManager class. This section lists and describes the content of Access Configuration and Access Policy classes. It describes the Test classes that you use to perform access tests against one or more users whose information you specify. The section concludes with a discussion of the ObAMException class, which is used for error handling.
- "Access Management API Classes" on page 435, which defines the classes that make up the Access Management API and gives examples of the class definitions in the Java programming language, managed code for C++, and, in some cases, the C programming language. For some classes, code samples are included to show how objects of the class are created and how methods of the class are used to act on those objects.

About the Access Management API

The Access Management API provides an interface which allows custom applications to access the authentication, authorization, and auditing services of the NetPoint Access Server to create and modify Access System policy domains and their contents. Before using the Access Management API, see the *NetPoint 7.0 Administration Guide Volume 2*. To better understand the functions provided by the Access Management API, explore the Access Manager GUI.

The NetPoint Access Management API provides Java, C, and managed code bindings for classes which you can use to instantiate these objects:

Access Policy Objects—These objects are used for data that is part of policy domains, policies, access conditions, audit rules and other policy domain content that Access Administrators ordinarily configure through the Access Manager.

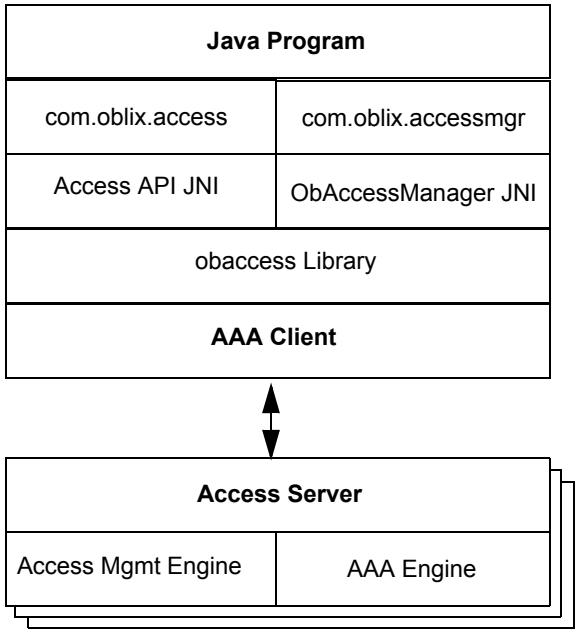
System Configuration Objects—These objects are used for data that is part of Authentication Plug-ins, Authentication and Authorization Schemes, and other policy domain content that the Access System Administrator ordinarily provides through the Access System Console > Access System Manager.

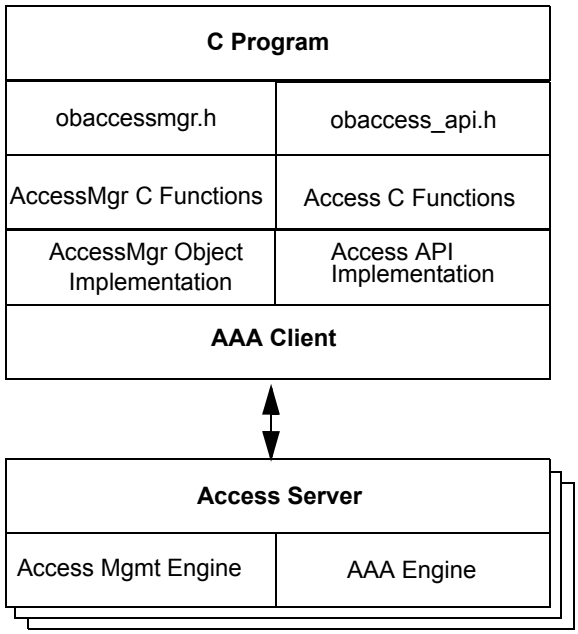
Some system configuration objects, including authentication and authorization schemes, the master audit rule, and host identifiers, are exposed by the API as read-only objects. You can use these objects within your programs just as you would use their corresponding Access System components. For example, just as you cannot modify an authentication scheme from the Access Manager, you cannot modify an authentication scheme object using the Access Management API.

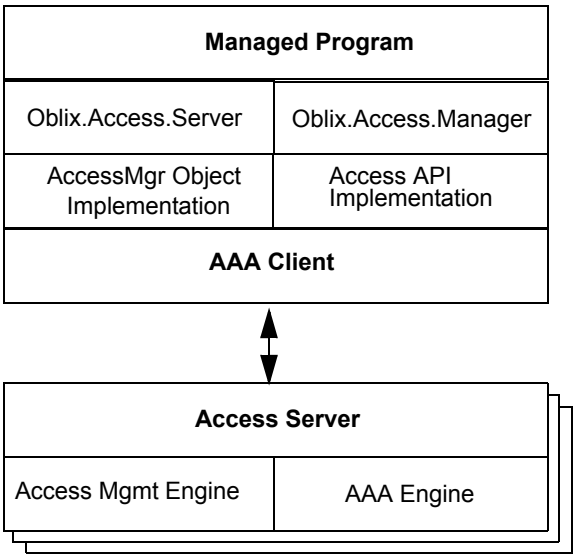
Access Server Connections—The ObAccessManager class represents one or more connections to Access Servers hosting the Access Management Service. An application uses ObAccessManager methods to send requests to get and set policy objects and to get configuration objects (setting of configuration objects is not supported in this release of NetPoint).

The following two diagrams show the structure of the Java and C bindings for the Access Management API.

In both cases, communication between the Access Client and the Access Server is provided by the NetPoint Transport Protocol.


Java Program
com.oblix.access
com.oblix.accessmgr
Access API JNI
ObAccessManager JNI
obaccess Library
AAA Client
Access Server
Access Mgmt Engine
AAA Engine


C Program
obaccessmgr.h
obaccess_api.h
AccessMgr C Functions
Access C Functions
AccessMgr Object Implementation
Access API Implementation
AAA Client
Access Server
Access Mgmt Engine
AAA Engine



Notes on Managed Code

In the following sections, managed code examples are depicted as managed C++. However, managed code can be written in any language supported by the Microsoft .NET managed code framework.

The managed code API resembles very closely the C and Java APIs. Classes and method names between the APIs are very similar. Here are some of the key differences:

- The C enumerators used to specify various conditions are wrapped by managed classes.
- Certain method names have been changed to match the naming conventions for properties. A property is similar to a member variable in that it allows the user to read and write values to an object. However, the property is implemented via get and set methods.

All managed class names are appended with the Mgd tag. Managed helper classes are documented in “Managed Helper Classes” on page 669.

Development Environment

The Access Management API is a subset of the Access Server Software Developer's Kit (SDK), which also supports the Access Server API. This section describes the structure of the installed Access Server SDK directory and subdirectories, and describes the initial steps in the build process. See "Installing the Access Server SDK" on page 1140 for installation process details.

Installation Location

The Access Server SDK is packaged as a product distinct from the NetPoint Access Server itself, and must be installed in a separate step. Oblix recommends that you install the SDK on the same system that will run the application you want to build using the API. Note the location where the SDK is actually installed. The installation location is needed as part of various path definitions to be set up later, and is called *SDK_install_dir* in the rest of this chapter. For example, for the Windows programming environment, you might specify the following path as the installation directory:

C:\NetPoint\AccessServerSDK

Note: NetPoint's *Access Server* is distinct from the Access Server SDK and installs at a different default location. See "Authentication Plug-in API" on page 517 and "Authorization Plug-in API" on page 567 for information about the APIs you can use to create custom authentication and authorization plug-ins.

Installation Content

Within the installation directory, the following directories and content are provided:

apidoc—Provides online documentation for the Java version of the API in HTML format. The documentation describes two distinct packages. First of these is com.oblix.access, which is used by both the Access Management API and the Access Server API; see "Building AccessGates with the Access Server SDK" on page 245. The other is com.oblix.accessmgr, which is used only by the Access Management API.

examples—Includes a sample build file and sample make script, and example applications in subdirectories. The subdirectory obaccess contains an example servlet, and prototypes for classes that can be used to extend the startup and shutdown classes for the server application.

include—Contains header files that define the classes, methods and functions that make up the C++ and C classes for the obaccess library. In particular, the obaccessmgr_api.h file contains the descriptions for the Access Management API C functions.

oblix—Holds these five subdirectories and subcontent:

apps—Contains the netlibmsg.lst file of messages available to the AccessGate if errors occur. This is located in the path common/bin. At the discretion of the AccessGate, these could be logged locally (if the AccessGate has been configured to maintain a log), displayed locally, or ignored.

config—Contains nothing, immediately following installation. When you run the configuration tool and set the AccessGate to run in simple or cert mode, configuration data for the AccessGate is stored here. This includes an encrypted password file and the random seed for the key used to do the encryption. If the connection to the Access Server is secured, then Public Key certificate data is also provided in this directory.

lib—Contains the Access Server SDK libraries and Java archive files that are built into the application.

- **Various library files**—Contain the libraries required by the API: obaccess.dll for NT and Windows 2000, obaccessmgr.so for Solaris and for AIX, and obaccessmgr.sl for HP-UX. Each of these libraries contains the complete implementations of the policy and configuration objects and the ObAccessManager. The ObAccessManager uses methods from the aaa_client library appropriate to each platform to communicate with one or more Access Server(s) hosting the Access Management Engine(s).

  The obaccess libraries also contain the policy objects, get functions for some of the configuration objects, and a subset of the ObAccessManager functions. The libraries also carry the native side of the JNI interface used by the Java binding.

- **jobaccess.jar**—Is the Java archive file for the API. It contains two packages, com.oblix.access and com.oblix.accessmgr. Some methods from com.oblix.access are used by the Access Management API to communicate with Access Servers.The com.oblix.accessmgr package contains all other Access Management API methods.

- **ObAccessGate.msg**—Provides the text of messages that the AccessGate may need to display.

- **ObAccessClient.lst file**—An example of the configuration file that is used by the AccessGate. For details on the contents of this file and modifying an AccessGate, see the *NetPoint 7.0 Administration Guide Volume 2*.

orig—Contains information created during the installation of the SDK; this directory can be ignored.

tools—Contains three significant directories:

- **configureAccessGate directory**—Contains the tool used to configure the AccessGate and the messages needed by it.
- **The migrationtools directory**—Contains information useful for migrating earlier versions of the API.
- **The openssl directory—**Contains a tool and certificate file used to configure the AccessGate for simple or cert mode operation.

samples—Contains versions of the standalone "Access Test" application, in Java, C, C++ as well as example Access Management API applications. You can use these applications to become familiar with the SDK build process before going on to more complex applications.

Note: Do *not* change the relative locations of the subdirectories and files within the SDK once it is installed. Doing so may prevent an accurate build and proper operation of the API.

About Building an AccessGate

Before you build your application that implements an AccessGate using the Access Management API, you must set or modify certain environment variables.

Environment Variables

To ensure that API components can be located, you must set certain environment variables before you compile the application that uses the API. See the discussion in “Setting Environment Variables” on page 259.

Build Process

Sample files to create servlets and build JAR files are provided at different levels within the examples directory. See “Installation Content” on page 407.

Configuration File

The AccessGate you build with the API *must* be properly configured or it will not be able to interact with the Access Server. You configure the AccessGate by defining the content of a configuration file on the system where the AccessGate is running. A discussion of the content and use of this file, and modifying an AccessGate, is in the *NetPoint 7.0 Administration Guide Volume 2*.

Coding With the Access Management API

The Access Management API provides functionality equivalent to that of the NetPoint Access System's standard Access Manager GUI. The API includes classes to represent and manage the same data objects managed through these GUIs.

Access System Configuration Objects—Represent data managed through the Access System Console, including authentication and authorization schemes, host identifiers, resource types, and the master audit rule. Discussion of these objects starts at page 434.

Access Policy Objects—Represent the data managed through the Access Manager GUI, including policy domains with all their subordinate information: policies and authentication, authorization, audit, and admin rules, and authorization expressions. Discussion of these objects begins on page 445.

The ObAccessManager Class—Provides a way to connect to the Access Server and to create, retrieve, modify, and delete Access Policy objects, and to retrieve Access System Configuration objects. This class also provides methods to test access policies. Discussion of this class starts at page 422.

Exception Classes—Provide methods for capturing and presenting errors that may occur. Discussion starts at page 508.

API Conventions

The Access Management API follows the object-oriented programming concept, in which classes define objects and methods, and the methods within the class operate upon the content of the objects.

The Access Management API classes are implemented in the Java programming language and as managed code classes. The Access Management API also includes an equivalent C programming language implementation. Definitions of the classes and functions for the supported languages are provided in the "Access Management API Definitions" on page 621.

Note: In the examples of this chapter, the managed code is appropriate for managed C++. However, keep in mind that you can write managed code using any language supported by the Microsoft .NET framework for managed code, including Visual Basic, C#, and C++.

Programmatic and Implementation Conventions

In Java, methods are called by reference. The application knows what class the object was created under and therefore which method to use. For example, to first create an authentication rule named myauthrule, and then set a description for it, you would write the following code in Java:

```
ObAMAuthenticationRule myauthnrule =
        new ObAMAuthenticatonRule[];
myauthnrule.setDescription("The description text");
```

For managed code, how the methods are used depends on the implementation language. For example, to create an authentication rule and a description for it, you would write the following code:

```
ObAMAuthenticationRuleMgd *myauthnrule = new
new ObAMAuthenticatonRule();
myauthnrule->Description="The description text";
```

The C implementation mimics the Java operation by the extensive use of opaque pointers to an underlying set of structures. (Fortunately, you need to deal with only the top layer of these.)

Classes and objects are referenced by pointers to their structures. When an object is created, the returned value is a pointer to a structure defining that object. The pointer type is always the object's class name with the text `_t` appended. When you use a C function to change a member of the structure for the object, you include the pointer as an argument to indicate which object you are referring to.The pseudo-method name used to make the change is precisely defined by preceding it with the class name of the object whose structure is being modified. In C, to create an Authentication Rule named myauthrule and set a description for it, you would write the following code:

```
ObAMAuthenticationRule_t myauthnrule;
myauthnrule = ObAMAuthenticationRule_new();
ObAmAuthenticationRule_setDescription(
        myauthnrule, "The description text")
```

Naming Conventions

To the greatest extent possible, classes, objects, and methods all have names that describe what they contain or what they do. For example, the class ObAMAuthenticationRule provides methods for manipulating Authentication Rule objects. In this class, the method setDescription is used to add a description for an Authentication Rule object. The ObAMAuthenticationRule class can have associated with it actions to be taken depending on the outcome of the authentication process.

The ObAMAuthenticationRule class inherits automatically the abstract class ObAMObjectWithActions class, and it uses its methods of this class for dealing with actions. As do the methods of most classes, those of the ObAMObjectWithActions class also have meaningful names. Any class that uses actions inherits the ObAMObjectWithActions class. For this reason, the names of the methods of the ObAMObjectWithActions class are general. For example, this class includes a method called addActionofType.

Creating New Objects

Each class has a method which acts as a constructor to build an empty object of the class. Java uses its built-in new() method to do this and the new object is manipulated by reference. The C version of the constructor is named after the object to be created with the text _new appended. The constructor returns a pointer, whose type is the object's class name with _t appended.

For example, to create a new Policy object in Java you would write the following code:

```
ObAMPolicy mypolicy = new ObAMPolicy();
```

In managed code, to create a new Policy object in managed C++, you would write the following code:

```
ObAMPolicyMgd * mypolicy = new ObAMPolicyMgd();
```

and in C you would write the following code:

```
ObAMPolicy_t mypolicy;
mypolicy = ObAMPolicy_new();
```

Copying Existing Objects

Each class has the equivalent of a copy constructor that makes a copy of an existing object of the same class, including all its members. In Java the `clone()` method is used to copy an object, and the new object is manipulated by reference. Be sure to cast the cloned object to the right class.

The equivalent C function is named after the class, with the text, `_copy`, appended. The function returns a pointer to the new object. The function takes an argument that is the name of the object to be copied.

For example, in Java to make a copy of an existing action object called action1 and to call the new object action2, you would write the following code:

```
ObAMAction action2 = (ObAMAction) action1.clone();
```

In managed C++, you would write the following code:

```
ObAMActionMgd *action2=(ObAMActionMgd *) action1->Clone();
```

In C you would write the following code:

```
ObAMAction_t action2;
action2 = ObAmAction_copy(action1);
```

About Cloning Objects Explicitly

To clone an object, you must copy it explicitly. For example, you cannot set a new value for an existing object and presume that by adding the object to the policy domain it will be cloned. The following snippet of code attempts to implicitly clone a resource object called resource1.

```
ObAMResource resource = new ObAMResource();
resource.setResourceType("http");
resource.setHostID("host1");
resource.setURLPrefix("/myresource1");
domain.addResoure(resource);
resource.setURLPrefix("/myresource2");
domain.addResource(resource);
```

The Access Server interprets the code as an attempt to change the URL of the first resource from /myresource1 to /myresource2. The resource object called resource has already been added to the policy domain. Rather than clone that resource, the Access Server interprets the following line of code as instruction simply to change the URL of the resource:

```
resource.setURLPrefix("/myresource2");
```

When the last line of code attempts to add the same resource again to the domain, the Access Server reports that it already exists.

The correct way to accomplish the intended task is to create a copy of the object and change the current values of any of its properties as required. Here is an example of how you might clone an object. This snippet of code attempts to add two resources to a policy domain. The second resource to be added is of the same type and has the same hostID as the original one.

```
ObAMResource resource1 = new ObAMResource();
resource1.setResourceType("http");
resource1.setHostID("host1");
resource1.setURLPrefix("/myresource1");
domain.addResource(resource1);
ObAMResource resource2 = (ObAMResource) resource1.clone();
resource2.setURLPrefix("/myresource2");
domain.addResource(resource2);
```

The programmer wants to add a resource of type http whose host ID is host1 and whose URL is /myresource1 to the policy domain called domain. The code creates an object of type ObAMResource called resource1, and sets its values. Then it adds resource1 to the policy domain (domain).

After adding the first resource to domain, the programmer wants to add another resource of the same type (http) that is on the same host (host1). The URL for that resource is /myresource2.

For this purpose, the developer clones resource1, creating an object called resource2, which inherits all of the properties of resource1.The only property the developer changes is the URL. The developer sets the URL for resource2 to /myresource2.

Deleting Objects

Each class provides the equivalent of a destructor that deallocates memory used for an object, including *member objects*.

Note: *Member objects* are other objects that the primary object contains. For example, AdminRule objects contain Identity objects.

Java does not actually use destructors but rather garbage-collects objects that are no longer referenced.

Managed code is similar to Java in its use of a garbage collector. When an object goes out of scope and it is no longer referenced, it is automatically deleted.

In C, a specific function is used, which is named after the class to which the object belongs and which has appended to it _delete. The function takes one argument, a pointer to the pointer to the object to be deleted. The multiple levels of pointing are used by the function first to deallocate memory for the object and then to reset the object pointer to NULL after the object has been deleted.

For example, in C to deallocate an existing action called actionname from the ObAMPolicy class, you would write the following code:

```
ObAMPolicy_delete(&actionname);
```

Managing Data for Single-Valued Object Members

Objects can contain members having single-valued data members, and they can contain arrays of data or objects. For example, a policy domain can contain arrays having multiple resources (an array of objects) and multiple operations (an array of strings) for each resource type of a policy.

This section explains how to manage objects which contain single-valued data members.

Setting Data for Single-Valued Object Members

Each class has methods beginning with those that set the content for single-valued members of objects in the class. For these cases, the data is a string, integer, Boolean flag, or a reference to an object. The method name is always set*widget*, where *widget* is the type of information to be set. In Java, the new data for the member is set by reference, and old data for the member may become eligible for garbage collection.

In the managed code API, any set method that takes only one value is used as if you were directly accessing the data member. For example, to set the description in the ObAMAuthenticationRuleMgd class, the method is treated as if it were a data member:

```
myauthnrule->Description = "The description text";
```

C is a bit more complicated. The first use of the set method stores a reference to the source object. Subsequent use of set for the same source object copies from the source to the receiving object member. The C function always takes two arguments. The first argument is always the name of the object for which data is being set. If the function is inserting a value, then the second argument is that value. If the set is inserting a reference to an object then the second argument is the reference.

For example, the code to set the value for a resource type (a string) and to add an authentication rule called myauthrule1 (an object) in an existing policy called mypolicy looks like this for Java:

```
mypolicy.setResourceType("http");
mypolicy.setAuthenticationRule(myauthrule);
```

And like this for managed C++:

```
mypolicy->ResourceType="http";
mypolicy->AuthenticationRule=myauthrule;
```

And it looks like this in C:

```
ObAMPolicy_setResourceType(mypolicy, "http");
ObAMPolicy_setAuthenticationRule(mypolicy, myauthrule);
```

Getting Data for Single-Valued Object Members

Each class has `get` methods that are used to retrieve the content for single-valued data members. For both Java and C, the method returns NULL if no value has been set. The function name is always get*widget*, where *widget* is the type of information to be retrieved. In Java, the object is identified by reference. In C, the object name is provided as an argument to the function.

For example, the methods to get the value for the policy name and to get a pointer to an authentication rule object within a policy object called mypolicy are represented this way in Java:

```
namevar = mypolicy.getName();
rule1 = mypolicy.getAuthenticationRule();
```

and in this way in managed C++:

```
namevar = mypolicy->Name;
rule1 = mypolicy->getAuthenticationRule;
```

and in this way in C:

```
const char *namevar;
namevar = ObAMPolicy_getName(mypolicy);
ObAMAuthenticationRule_t rule1;
        rule1 = ObAMPolicy_getAuthenticationRule(mypolicy);
```

Managing Arrays

An object member with multiple elements each having its own value is organized as an array indexed from 0. In order to retrieve information for an element of the array, you need to ask for it based on its position in the array specified as an index.

Objects can contain arrays of either values or other objects. This section describes the following topics, which discuss the kinds of functions you can perform on array members of objects:

About Keys

Many objects contain *key* members. Key members are useful in cases where an object contains an array of member objects. The key concept allows methods to search for the member object in the array before performing the requested operation.

Note: No set methods are provided for object members which are arrays. Instead, you must replace the old value with a new one. For example, to change the value of an operation in an array of operations for a Policy object (of the policy class), you must remove the old value and add the new value in two steps.

Adding Data to Arrays

Objects that contain arrays have methods that allow you to add a new member to the array. If the array is an array of values, then the value is appended to the array and the number of elements in the array increases by one. If the array is an array of objects, then a reference to the new object is added, and the number of elements in the array increases by one. The method name is always add*widget*, where *widget* is the type of information being added. In Java, managed code, and C, the method takes an argument, either the value to be added or the name of the object being added.

For example, given a Policy object called mypolicy, to add another Operation (a string) or to add another resource (an object), the Java code looks like this:

```
mypolicy.addOperation("GET");
mypolicy.addResource(myresource);
```

The managed C++ code looks like this:

```
mypolicy->AddOperation="GET";
mypolicy->AddResource=myresource;
```

The equivalent C code requires an additional argument, the object to which the value or object is being added, and looks like this:

```
ObAMPolicy_addOperation(mypolicy, "GET");
ObAMPolicy_addResource(mypolicy, myresource);
```

Modifying Data for Objects in Arrays

Objects can contain arrays of member objects whose values you can modify. You can do this by getting the member from the object, changing its data, and then storing it back in the object. A more facile way to do this is to create an empty member object, fill in the data that needs to be modified, and then use the modify method to overwrite the existing version of the member object in the array of objects.

To modify data for objects in arrays

1. Create an empty member object (of an array of objects), all of whose members will initially be filled with `NULL`.
2. Set the value for the *key* member in the empty member object to match the value for the *key* member in the member object to be overwritten.

   Note: Refer to the tables for each object type to see which is the key member.

3. Set values in the members of the member object that you want to change. Leave the values of members that are not to be changed set to `NULL`.
4. Use the modify method to overwrite the old member object. The function name is always modify*widget*, where *widget* is the type of object being modified. Modify takes one argument, the name of the source member object. Based on the value of the key member, modify locates the matching member object. Where the source member object members are set to NULL, those values in the receiving member object will be unchanged. Where values were specified for members, those values will be changed in the receiving member object.

For example, suppose you want to change the description for a resource whose resource type member is EJB, and the resource is one of the resource members of an array of resource objects for the Policy object mypolicy.

To accomplish this in Java, you would write the following code:

```
ObAMResource exmplResr = new ObAMResource();
exmplResr.setResourceType("EJB");
exmplResr.setDescription("The New Description");
mypolicy.modifyResourceType(exmplResr);
```

To accomplish this in managed C++, you would write the following code:

```
ObAMResource *exmplResr
        exmplResr = new ObAMResource();
exmplResr->ResourceType="EJB";
exmplResr->Description="The New Description";
mypolicy->ModifyResourceType=exmplResr;
```

To do the same in C, you would write the following code:

```
ObAMResource_t exmplResr;
        exmplResr = ObAMResource_new();
ObAMResource_setResourceType(exmplResr, "EJB");
ObAMResource_setDescription(exmplResr,"The New Description");
ObAMPolicy_modifyResourceType(exmplResr, exmplResr);
```

Getting a Count of Members in an Array

A member with multiple elements each having its own value is organized as an array, indexed from 0. Your application can obtain the value of any element of an array. To do so, you use the get method, which returns the value for the requested element. However, to retrieve information for an element of the array, you need to ask for it by its position in the array.

Before your application calls the get method to get the value of an element of an array, you must know how many members the array contains. You use the getnumberof*widgets* method for this purpose, where *widget* is the type of information the array contains.

The Java version of this method provides the object name by reference, and has no arguments. The C version takes a single argument, the name of the object.

To get the number of resources or the number of operations in a Policy object for a policy domain named mypolicy, example Java code is:

```
mypolicy.getNumberOfResources();
mypolicy.getNumberOfOperations();
```

The equivalent managed C++ code is:

```
int resNumber = mypolicy->NumberOfResources;
int operNumber = mypolicy->NumberOfOperations;
```

The equivalent C code is:

```
ObAMPolicy_getNumberOfResources(mypolicy);
ObAMPolicy_getNumberOfOperations(mypolicy);
```

Getting Data for Elements of Arrays

After your application has called the getNumberOf*widgets* method to obtain a count of the number of elements of an array, you can get the value of an element of the array. You pass the appropriate get method the index to the element of the array whose value you want returned.

Note: Be careful not to confuse these `get` methods with the ones used to extract values for single-valued data. You can tell which to use from the member type. If the member type is *array*, then the get is asking for the value of an element in the array and you must pass the method the index of the element.

The Java methods take one argument, the index. The C functions take the index as the second argument. For C functions, the first argument is the name of the object holding the member from which the data is being retrieved.

For example, from the policy domain called mypolicy, to get the value for one operation in an array of operations or for one policy in an array of policies, the Java code is:

```
mypolicy.getOperation(myindex);
mypolicy.getPolicy(myindex);
```

The managed C++code is:

```
System::String *operation = mypolicy->getOperation(myindex);
ObAMPolicyDomainMd *policy = mypolicy->getPolicy(myindex);
```

The C code looks like this:

```
ObAMPolicyDomain_getOperation(mypolicy, myindex);
ObAMPolicyDomain_getPolicy(mypolicy, myindex);
```

Removing Data from Arrays

Objects that contain arrays of values or of member objects have methods that allow you to remove members of the array. To do this, you specify a value to be matched. If the array is an array of values, then the match is done on one of those values. If the array is an array of member objects, then the match is done on the value stored for the *key* member of the object.

Note: Refer to the tables for each object type to see which is the key member.

Both Java and C take the value to be matched as an argument. For C, this is the second argument; the first is the object from which the value is to be removed.

For example, if you want to remove the operation GET from the Policy mypolicy, the Java code is:

```
mypolicy.removeOperation("GET");
```

The managed C++ code is:

```
mypolicy->removeOperation="GET";
```

The C code:

```
ObAMPolicy_removeOperation(mypolicy, "GET");
```

Using setIDFrom

Every object received from the Access Server contains within its structure a unique identifier, in addition to its name or other key information. Another object of the same type can have its ID set to the same value. A minor change can then be made to the second object, and modify can be used to write just the change back to the original. For simple changes, this is superior to using copy, because no processing time needs to be spent to copy data that will not be changed.

Both Java and C take the name of the object whose ID is to be copied as an argument. For C, this is the second argument; the first is the object whose ID is to be set.

For example, if you want to set the ID for a Policy called workingpolicy to be the same as the ID for an existing Policy called oldpolicy, the Java code is:

```
workingpolicy.setIDFrom(oldpolicy);
```

The equivalent managed C++ code is:

```
workingpolicy->IDFrom=oldpolicy;
```

The equivalent C code is:

```
ObAMPolicy_setIDFrom(workingpolicy, oldpolicy);
```

Using Enumerations

Integer constant values are used to represent sets of predefined valid inputs for certain object members. In this chapter, these are listed as part of the description for the object. For example, the ObAMAction class contains an enumeration with three values:

```
UNDEFINED = 0;
FIXEDVALUE = 1;
ATTRIBUTE = 2;
```

It is possible for a programmer to pass an incorrect enumerated value to an object within the Access Management API. To catch this kind of error, the Java binding checks the enumerated values at run time and throws an ObAMException if an incorrect value is passed.

Managed code uses the ObAmAction_ValueTypeMgd class with the following methods:

```
isUndefined
isAttribute
isFixedValue
setUndefined
```

The managed code has wrapped all the needed enums as objects. Each enum has the prerequisite getter and setter methods. For example, the ObAMAction_ValueType enum has been wrapped by the object ObAMAction_ValueTypeMgd class. To create this class and set the value type as FixedValue, you would write the following code:

```
ObAMAction_ValueTypeMgd valueType = new
ObAMAction_ValueTypeMgd();
valueType->setFixedValue();
if (valueType->isFixedValue == true) {
     <do something here>
}
```

The Access Manager code within the Access Server also checks input enumerated values sent to it for an improperly formatted request. If an error exists, the ObAccessManager method that originated the request receives the error and throws the exception.

ObAccessManager Class

ObAccessManager objects are the main objects used by the Access Management API to interact with the Access Server. Each ObAccessManager object represents one or more connections to Access Servers to which requests can be sent to get or set policy or system configuration objects. ObAccessManager objects support methods to perform the following functions:

- Establishment of Access Manager object(s)
- Connection to the Access Server
- Listing of top level existing objects, by using a special form of `get`
- Setting of values for new policy domains or changing values for existing policy domains, by using a special form of `set`
- Testing access to the Access Server

Methods to Handle AccessManager Objects

The following methods create a named AccessManager object and define the login information for the user. They are:

- A constructor for the AccessManager Object

  These use the standard constructor syntax described earlier. See “Creating New Objects” on page 412.

  For Java, to create a new AccessManager object myam, you would write the following code:

  ```
  ObAccessManager myam = new ObAccessManager();
  ```

  For managed C++, to create a new AccessManager object myam, you would write the following code:

  ```
  ObAccessManagerMgd myam = new ObAccessManagerMgd();
  ```

  For C, to create a new AccessManager object myam, you would write the following code:

  ```
  ObAccessManager_t myam;
  myam = ObAccessManager_new();
  ```

- A destructor for the AccessManager Object

  These processes use the standard destructor syntax described earlier. See “Deleting Objects” on page 414. For Java, the AccessManager objects are removed when they are no longer referenced.

  For managed C++, to remove the AccessManager object called myam, you let it go out of scope. Garbage collection will take care of removing the object.

  For C, to remove the AccessManager object called myam, you would write the following code:

  ```
  ObAccessManager_delete(&myam);
  ```

- The setAdmin method

  This method specifies the userid and password for an administrator for whom requests will be authorized. If the administrator does not exist or the password is not correct, ObAccessManager throws an exception with code ADMIN_LOGIN_FAILED.

  Example code for this in Java is:

  ```
  myam.setAdmin(“A. Loomis”, “ALoomisPassword”);
  ```

  Example code for this in managed C++ is:

  ```
  myam->setAdmin(“A. Loomis”, “ALoomisPassword”);
  ```

  and in C is:

```
ObAccessManager_setadmin_password(myam,"A. Loomis",
        "ALoomisPassword");
```

- The setCacheUpdates method

  This method sets a Boolean flag to specify whether or not Access Server caches are to be updated for each ObAccessManager `set` request.

  Example code for this in Java is:

  ```
  myam.setCacheUpdates(true);
  ```

  Example code for this in managed C++ is:

  ```
  myam->CacheUpdates=true;
  ```

  and example code for this in C is:

  ```
  ObAccessManager_setCacheUpdates(myam, 1);
  ```

Connection Methods

The Access Management API uses the Access Server API connection methods from the ObConfig class to establish one or more connections with the Access Server. The methods listed below are described in more detail in "ObConfig" on page 280.

- The initialize method

  The initialize method takes an optional argument that specifies the (local) directory path where the Access Management API has been installed. If this argument is not given, ObAccessManager will get the directory path from the local environment variable OBACCESSMGR_INSTALL_DIR.

  The initialize method creates a connection to an Access Server specified in the installation configuration file. See the discussion of this file and modifying an AccessGate in the *NetPoint 7.0 Administration Guide Volume 2*. This connection will be maintained until one of the following occurs:

  - The shutdown method is called.
  - The connection is broken, in which case ObAccessManager will attempt to re-establish the connection to the same or another configured Access Server.
  - The configured (API) client session timeout is reached.

    If no connection can be established to any configured Access Server, ObAccessManager will throw an ObAMException with code CANNOT_CONNECT.

  Assuming that the local environment variable is set to specify the installation directory location, here is the Java call:

  ```
  Obconfig.initialize();
  ```

  For C it is:

```
Obconfig_initialize(myam);
```

For managed code, name space for the common classes is:

```
Oblix::Access::Common
```

For managed code, the initialize method is a static method of the ObConfigMgd class:

```
ObConfigMgd.initialize();
```

- The shutdown method

  The shutdown method closes all connections and deallocates resources used by the Access Client.

  Example coding for this in Java is:

  ```
  Obconfig.shutdown();
  ```

  In managed C++ it is:

  ```
  ObConfigMgd.shutdown();
  ```

  In C it is:

  ```
  Obconfig_shutdown(myam);
  ```

- The getSDKVersion method

  The getSDKVersion method returns the version of the Access Server SDK under which the Access Management API was built.

  To make this call in Java, you would write the following line of code:

  ```
  theSDKv = Obconfig.getSDKVersion();
  ```

  To make this call in managed C++, you would write the following line of code:

  ```
  string theSDKv = ObConfigMgd.SDKVersion;
  ```

  and to make the call in C:

  ```
  char * theSDKv;
  theSDKv = Obconfig_getSDKVersion(myam);
  ```

- The getNAPVersion method

  The getNAPVersion method returns the version of the NetPoint Access Protocol used to communicate with Access Servers as defined by the Access Server.

  To make this call in Java, you would write the following line of code:

  ```
  theNAPv = Obconfig.getNAPVersion();
  ```

  To make this call in C++, you would write the following line of code:

```
theNAPv = Obconfig.NAPVersion;
```

and in C is:

```
char * theNAPv;
theNAPv = Obconfig_getNAPVersion(myam);
```

Get Methods

The ObAccessManager get methods return one or more policy domains or configuration objects from the policy directory via the Access Server. The ObAccessManager get methods allow the developer to get information of varying degrees of complexity for the following objects:

- AuthenticationScheme
- AuthorizationScheme
- HostIdentifier
- MasterAuditRule
- PolicyDomain
- ResourceType

The get methods return an array of objects dependent on the values passed to the method for the following arguments:

- `matchName`: selects the objects; if NULL, returns all objects.
- `matchCriterium`: specifies how objects are to be selected by the matchName.

For example, here are the prototypes of the get method for policy domains in the Java and C programming languages.

Java

```
public PolicyDomain[] ObAccessManager.getPolicyDomains(
        int responselength, String matchName,
        int matchCriterium)throws ObAMException;
```

C

```
ObAMArrayOfPolicyDomains_t ObAccessManager_getPolicyDomains(
        ObAccessManager_t am,
        ObAccessManager_ResponseLength responseLength,
        const char *matchName,
        ObAccessManager_MatchCriteria matchCriterium);
```

The arguments allow the developer to control the precision of the information returned for each object.

- **am**—For the C programming language, the name of an ObAccessManager object created using the ObAccessManager_new function.
- **responseLength**—Specifies how much information about the objects is to be returned.
  - There are three levels of increasing response complexity: MIN, MID, and MAX. If responseLength is omitted, the default is MIN.
  - Each get method knows implicitly which data items are to be returned for each level, as indicated by the following table.
  - Placeholders are returned for all possible data in the structure(s) defining the object. For items for which they do not return values, MIN and MID return NULL pointers/references.

  Enumerated values for specifying responseLength are:

```
MIN = 0
MID = 1
MAX = 2
```

Object Type	MIN	MID	MAX
`Authentication Scheme`	-Name -Description (In effect, List all Authentication Schemes)	Same as MIN values	-Level -Challenge method -Challenge parameters -SSL required -Challenge redirection -Plug-ins
`Authorization Scheme`	-Name -Description (In effect, list all Authorization Schemes)	Same as MIN Values	-Shared Library - User parameters -Required parameters -Optional parameters
`HostIdentifier`	-Name -Description (In effect, list all host identifiers)	Same as MIN Values	-Hostname variants
`MasterAudit Rule`	All of the object data; no criteria may be applied	Same as MIN	same as MIN

`PolicyDomain`	-Name -Description -Resource types -URL prefixes -Enabled/ disabled flag (In effect, show “My Policy Domain”)	-Domain names -Policy names -Default rule names -URL prefixes (In effect, show Search results for Policy Domains)	-Complete default rules -Complete policies -Complete admin rules
`ResourceType`	-Name -Display Name	Same as MIN Values.	-caseSensitive Matching operations

- **matchName**—along with matchCriterium, is used to select specific objects for which values are to be returned. The value of matchName is compared to the value of the name member for each candidate object. If matchName is omitted, all objects of the class are retrieved.
- **matchCriterium**—specifies how object names are to be compared with matchName. The comparison is applied after the matchName has been applied. The choices are as described in the following table. If matchCriterium is omitted, the default is `EQUALS = 0`.

  Enumerated values for matchCriterium are:

```
EQUALS =              0
CONTAINS =            1
CONTAINS_IN_ORDER =   2
BEGINS_WITH =         3
```

```
ENDS_WITH =           4
```

Parameter	Meaning
EQUALS	The object name matches exactly the value provided.
CONTAINS	The object name contains the exact string specified. It may be embedded within other characters.
CONTAINS_IN_ORDER	The object name matches in the string the characters specified, in the specified order, but not necessarily as one contiguous string. For example, “123” would find a match in both “01234” and “102030”.
BEGINS_WITH	The object name begins with the exact value provided, however long it may be.
ENDS_WITH	The object name ends with the exact value provided, however long it may be.

Note: Security. The getPolicyDomains method returns only those policy domains and policies within domains for which the Access Manager administrator has at least basic admin rights. All other ObAccessManager get methods require that the admin user be a Master Access Administrator.

Managed Code Form

For managed code, the get method returns an array of objects for the class the method belongs to.

```
ArrayList*ObAccessManagerMgd::getPolicyDomains(
       ObAccessManager_ResponseLengthMgd *responseLength,
       System::String *matchName,
       ObAccessManager_MatchCriteriaMgd *matchCriterium);
```

To set the arguments to the AccessManagerMgd get method, you can use the ObAccessManager_ResponseLengthMgd and the ObAccessManager_MatchCriteriaMgd classes.

Class ObAccessManager_ResponseLengthMgd

To set the responseLength argument for the AccessManagerMgd get method, you can use the ObAccessManager_ResponseLengthMgd class. This class provides a wrapper around the enumeration ObAccessManager_ResponseLength.

The ObAccessManager_ResponseLengthMgd class includes get and set methods. The methods you use to set values are shown here.

```
ObAccessManager_ResponseLengthMgd();
```

```
__property void set_Value(
        ObAccessManager_ResponseLength value);
void setMin();
void setMid();
void setMax();
```

where:

setMin() is equivalent to enum value MIN

setMid() is equivalent to enum value MID

setMax() is equivalent to enum value MAX

Class ObAccessManager_MatchCriteriaMgd

To set the MatchCriteria argument for the AccessManagerMgd get method, you can use the ObAccessManager_MatchCriteriaMgd class. This class provides a wrapper around the enumeration ObAccessManager_MatchCriteria.

The ObAccessManager_MatchCriteriaMgd class includes get and set methods. Here are the methods you use to set values:

```
ObAccessManager_MatchCriteriaMgd();
__property void set_Value(
     ObAccessManager_MatchCriteria value);
void setEquals();
void setContains();
void setContainsInOrder();
void setBeginsWith();
void setEndsWith();
```

where:

setEquals() equivalent to enum value EQUALS

setContains() equivalent to enum value CONTAINS

setContainsInOrder() equivalent to enum value CONTAIN_IN_ORDER

setBeginsWith() equivalent to enum value BEGINS_WITH

setEndsWith() equivalent to enum value ENDS_WITH

Get Method Examples

To get minimum information for all policy domains whose names contain the string “oblix”, you would write the following Java code:

```
myam.getPolicyDomains(MIN, “oblix”, CONTAINS);
```

For managed C++ code, you would write the following code:

```
ObAccessManager_ResponseLengthMgd* len = new
ObAccessManager_ResponseLengthMgd();
```

```
ObAccessManager_MathCriteriaMgd *matchCriterium = new
ObAccessManager_MatchCriteriaMgd();
len->setMin();
matchCriterium->setContains();
policyDomains = myam->getPolicyDomains(
     len, "Oblix",matchCriterium);
```

For C, you would write the following code:

```
ObAccessManager_getPolicyDomains(myam, ObAccessManager_MIN,
     "oblix", ObAccessManager_CONTAINS);
```

Set Method

The API includes *one* ObAccessManager set method which can be used in the process of creating, modifying, and removing *policy domains only*. The definition for this method in the Java, Managed Code, and C programming languages is provided below.

Note: To set values for other high-level objects, you use methods specific to their content. These methods are described in the pertinent sections that follow.

Java

```
public void setPolicyDomain(ObAMPolicyDomain PDname, int setAction);
     throws ObAMException;
```

Managed Code

```
void setPolicyDomain(obAMPolicyDomainMgd
     *value, ObAccessManager_SetActionMgd *setAction);
```

C Form

```
void ObAccessManager_setPolicyDomain(
     ObAccessManager_t accessmanagerObjectName,
     ObAMPolicyDomain_t PDname,
     ObAccessManager_SetAction setAction);
```

For the C definition, *PDname* represents the name of a PolicyDomain. You use the *setAction* parameter to specify the kind of action to be taken in relation to the specified policy domain object. These are the possible actions:

- **Create**—A PolicyDomain object is to be created. If objects with the same name already exist, the object will be not be created and the `set` method returns the error code `EXISTING_OBJECT` along with the name of the existing object.
- **Modify**—An existing PolicyDomain object is to be modified. All specified data for the object is modified; undefined data is not affected. If a specified

object does not exist, the set method will return the error code `NO_OBJECT` and the name of the missing object.

- **Remove**—An existing PolicyDomain object is to be removed. Only the object name is relevant; any data provided along with the object is ignored. If the object does not exist, the set method returns the error code NO_OBJECT and gives the name of the missing object.

  Enumerated values for *setAction* are:

```
CREATE = 0
MODIFY = 1
REMOVE = 2
```

Note: Security. The ObAccessManager admin user must have basic admin rights for the policy domains in question to be able to call setPolicyDomain. If `setPolicyDomain` changes the delegate, grant, or basic admin rules of the domain, the admin user must have delegate or grant rights to do that, as appropriate. If the admin user does not have sufficient rights, the set method returns a `NOT_AUTHORIZED` error code.

For example, to create a new Policy Domain in Java, you would write the following code:

```
myam.setPolicyDomain(mypolicy, CREATE);
```

In managed C++, you would write the following code:

```
public void setPolicyDomain(ObAmPolicyDomainMgd *pdName,
        ObAccessManager_SetActionMgd *setAction);
```

using the ObAccessManager_SetActionMgd setCreate method. The ObAccessManager_SetActionMgd class includes the following methods for specifying the action:

```
void setCreate() = CREATE enum
void setModify() = MODIFY enum
void setRemove() = REMOVE enum
```

and in C you would write the following code:

```
AccessManager_setPolicyDomain(
        myam, mypolicy, ObAccessManager_CREATE);
```

Test Access Method

ObAccessManager provides a method to perform an access check, taking as its input an ObAMAccessTest object in which you specify a resource, an optional location, date and time, and a set of users. The testAccess method returns an ObAMAccessTestResults object that includes the applicable policy domain, policies, authorization expression, and determining authorization rules contributing to the outcome of evaluation of the authorization expression. The object also includes information specifying if access by each user is authorized.

Important: If you do not specify explicitly the users who are to be used in the access check test, all of the users in the LDAP directory will included in the test. You may want to perform an access test check using all of the users in the directory. However, if a test of this kind occurs unintentionally because you forgot to specify users, it can incur overhead and consume processing cycles, depending on the size of your LDAP directory.

In C, the caller is responsible for deleting the returned ObAMAccessTestResults object.

Note that this access test is similar but not identical to the ObUserSession.isAuthorized method of the Access Server API. See "ObUserSession" on page 277. The differences are:

- testAccess can test access for any user, while isAuthorized can only be used by an authenticated user
- testAccess can test access for any date and time, while isAuthorized always uses the current time.
- testAccess gets its information from the directory; the results are always current. isAuthorized gets its information from the Access Server cache, which may not be current.

The definition (not code example) for this method is:

Java

```
public ObAMAccessTestManaResults testAccess(
        ObAMAccessTest testname);
```

Managed C++ Form

```
public ObAMAccessTestResultsMgd
        *getTestAccess(ObAMAccessTestMgd *test);
```

C Form

```
ObAMAccessTestResults_t ObAccessManager_testAccess(
     ObAMAccessTest_t testname);
```

For example, presuming that the AccessTest object is already built then to do an Access Test in Java, you would write the following code:

```
myam.testAccess(myaccesstestobject);
```

In Managed C++, the code is:

```
myam.gettestAccess(myaccesstestobject);
```

and in C is:

```
ObAccessManager_TestAccess(myam, myaccesstestobject);
```

Access System Configuration Objects

Access System configuration objects represent certain basic object types for which information can be created, modified, deleted, and viewed through the Access System Console GUI. Because they underpin most of the access security supported by NetPoint, they cannot be changed using the Access Management API. However, these objects are used in creating policy domains and policies. With the exception of managed code supporting classes, you can get but not set values for the following objects included in this group:

Access Management API Classes

This section describes the Access Management API classes, including Java and managed code classes. In some cases, the Java class methods are followed by a list of the equivalent C programming language methods.

Also for each Java class, a table is provided showing the data members of the object manipulated by its methods and those of any classes it inherits. Here is how the table is organized:

- Each row of the table begins with the label used to identify the data member.
- The second column indicates the data type, and whether it is a single value or an array.
- The third column describes the range of legal values for the data, enumerated where appropriate
- The last column describes what each data item represents.

Some methods and functions use key data items in order to uniquely identify the object. Where there is only one key data item, that item is indicated as (key). Where two or more data items must be concatenated together to act as a key, they are indicated as (key1), (key2), and so on, in the order of concatenation.

Class ObAMHostIdentifier

Each object of the ObAMHostIdentifier class represents a configured host identifier.

Label	Type	Range	Description
Name (key)	A string Required	Any	The identifying name of the Host Identifier
Description	A string Optional	Any	An optional description
HostName	An array of strings Optional	Any	A set of hostname variations.

Class ObAMHostIdentifierMgd

(Managed Code). Each ObAMHostIdentifier object represents a configured host identifier. For information on the object components, see “Class ObAMHostIdentifier” on page 436.

Managed Code Form

```
// getters
__property System::String *get_Name();
__property System::String *get_Description();
__property int get_NumberOfHostnames();
System::String *getHostname(int index);
```

Class ObAMResourceType

Each object of the ObAMResourceType class represents a built-in or custom resource type.

Label	Type	Range	Description
Name (key)	A string Required	Any	The identifying name of the resource type, which allows its reuse
DisplayName	A string Required	Any	The displayed name
Case-Sensitive Matching	A Boolean flag Required	0, meaning no, or other, meaning yes	A flag indicating whether URL matching for the resource type is case-sensitive
Operations	An array of strings Optional	Any	A set of operations defined for the resource type

Java

```
public String getName();
public String getDisplayName();
public boolean getCaseSensitiveMatching();
public int getNumberOfOperations();
public String getOperation(int index);
```

Class ObAMResourceTypeMgd

(Managed Code). Each object of the ObAMResourceType class represents a built-in or custom resource type.

Label	Type	Range	Description
Name (key)	A string Required	Any	The identifying name of the resource type, which allows its reuse
DisplayName	A string Required	Any	The displayed name
Case-Sensitive Matching	A Boolean flag Required	True, false	A flag indicating whether URL matching for the resource type is case-sensitive
Operations	An array of strings Optional	Any	A set of operations defined for the resource type

Managed Code Form

```
// getters
__property System::String *get_Name();
__property System::String *get_DisplayName();
__property bool get_CaseSensitiveMatching();
__property int get_NumberOfOperations();
System::String *getOperation(int index);
```

Class ObAMAuthenticationScheme

Each object of the ObAMAuthenticationScheme class represents a configured authentication scheme. Every authentication rule must contain the name of an existing authentication scheme.

Label	Type	Range	Description
Name (key)	A string Required	Any	The identifying name of the Authentication Scheme
Description	A string Optional	Any	An optional description
Level	Numeric Required	Any	A level ranking the security of the scheme relative to other configured schemes
Challenge Method	A member of the enumerated list Required	Any one of the enumerated list: UNDEFINED=0 NONE=1 BASIC=2 X509=3 FORM=4 EXTERNAL=5	The method to be used to challenge the user for credentials.
Challenge Parameters	An array of strings Required	Any	A set of parameters to be used in the challenge
SSLrequired	A Boolean flag, Required	0, meaning no, or other, meaning yes	A flag indicating whether an SSL security connection is required for the challenge
Plugin	An array of Authentication Plugin objects Optional	See the class definition on page 441	An array of plug-ins to process the credentials to produce the authenticated user
Enabled	A Boolean flag,		A flag to enable and disable the authentication scheme

Java

```
public String getName();
public String getDescription();
public int getLevel();
public int getChallengeMethod();
public boolean getSSLrequired();
public String getChallengeRedirectURL();
public int getNumberOfChallengeParameters();
```

```
public String getChallengeParameter(int index);
public int getNumberOfPlugins();
public ObAMAuthenticationPlugin getPlugin(int index);
public boolean getEnabled();
```

Class ObAMAuthenticationSchemeMgd

(Managed Code). Each object of the ObAMAuthenticationSchemeMgd class represents a configured authentication scheme. See “Class ObAMAuthenticationScheme” on page 439 for details on the components of this object.

```
// getters
__property System::String *get_Name();
__property System::String *get_Description();
```

Class ObAMAuthenticationScheme_ChallengeMethodMgd

(Managed Code). This is a Managed Value Type class which defines the value types used by the ObAMAuthenticationSchemeMgd class. This class provides a wrapper around the enum ObAMAuthenticationScheme_ChallengeMethod. You can set the value using the setter methods, or you can use an ObAMAuthenticationScheme_ChallengeMethod value.

```
__property int get_Level();
__property ObAMAuthenticationScheme_ChallengeMethodMgd
        *get_ChallengeMethod();
__property bool get_SSLrequired();
__property bool get_Enabled();
__property System::String *get_ChallengeRedirectURL();
__property int get_NumberOfChallengeParameters();
__property int get_NumberOfPlugins();
System::String *getChallengeParameter(int index);
ObAMAuthenticationPluginMgd *getPlugin(int index);

// Get and set values
__property bool get_isUndefined();
__property bool get_isNone();
__property bool get_isBasic();
__property bool get_isX509();
__property bool get_isForm();
__property bool get_isExt();
__property ObAMAuthenticationScheme_ChallengeMethod get_Value();
__property void
set_Value(ObAMAuthenticationScheme_ChallengeMethod value);
void setUndefined();
void setNone();
void setBasic();
```

```
void setX509();
void setForm();
void setExt();
```

For managed code, the mapping between the enumerated list (the challenge method) and the managed code class is as follows:

```
ObAMAuthenticationScheme_ChallengeMethod:
setUndefined() = UNDEFINED
setNone() = NONE
setBasic() = BASIC
setX509 = X509
setForm() = FORM
setExt() = EXT
```

Class ObAMAuthenticationPlugin

Each object of the ObAMAuthenticationPlugin class represents an authentication plug-in that can be configured into an authentication scheme.

Label	Type	Range	Description
Order	An integer Required	Any	The order in which the plug-in is executed in the authentication scheme
Name (key)	A string Required	Any	The name of the plug-in (without the file extension), which is either a built-in plug-in name or the name of the custom plug-in library
Parameters	A string Optional	Any	A single string, of user defined parameters for the plug-in

Java

```
public int getOrder();
public String getName();
public String getParameter();
```

Class ObAMAuthenticationPluginMgd

(Managed Code). Each ObAMAuthenticationPlugin represents an Authentication Plug-in that can be configured into an authentication scheme. For details on the object components, see “Class ObAMAuthenticationPlugin” on page 441.

Managed Code Form

```
// getters
__property int get_Order();
__property System::String *get_Name();
__property System::String *get_Parameters();
```

Class ObAMAuthorizationScheme

Each object of the ObAMAuthorizationScheme class represents a custom authorization scheme.

Label	Type	Range	Description
Name (key)	A string Required	Any	The identifying name of the Authorization Scheme
Description	A string Optional	Any	A description to displayed for the scheme
Library	A string Required	Any	The name of the library file implementing the scheme
User Parameters	An array of strings Optional	Any	Parameters defined by the user
Required Parameters	An array of Parameter objects Optional	See the class definition on page 496.	Parameters built into the scheme and required for its correct operation, if any
Optional Parameters	An array of Parameter objects Optional	See the class definition on page 496.	Parameters with which the scheme will work, if they are provided

Java

```
public String getDescription();
public String getLibrary();
public int getNumberOfUserParameters();
public int getNumberOfRequiredParameters();
public int getNumberOfOptionalParameters();
public String getUserParameter(int index);
public ObAMParameter getRequiredParameter(int index);
public ObAMParameter getOptionalParameter(int index);
```

Class ObAMAuthorizationSchemeMgd

(Managed Code). Each ObAMAuthorizationSchemeMgd represents a custom authorization scheme. For details on components of this object, see “Class ObAMAuthorizationScheme” on page 442.

Managed Code Form

```
// getters
__property System::String *get_Name();
```

```
__property System::String *get_Description();
__property System::String *get_Library();
__property int get_NumberOfUserParameters();
__property int get_NumberOfRequiredParameters();
__property int get_NumberOfOptionalParameters();
System::String *getUserParameter(int index);
ObAMParameterMgd *getRequiredParameter(int index);
ObAMParameterMgd *getOptionalParameter(int index);
```

Java

```
public String getName();
public String getDescription();
public int getNumberOfHostnames();
public String getHostname(int index);
```

Class ObAMMasterAuditRule

A object of the ObAMMasterAuditRule class represents the master audit rule, which specifies global audit parameters and defaults to be used if there is no audit rule specified for a specific policy.

Label	Type	Range	Description
DateFormat	Enum Required	An integer from the following enumerated set of possible date formats: UNDEFINED=0 INTEGER = 1 MMDDYYYY=2 DDMMYYYY=3 ISO8601 = 4 YYYYMMDD=5 YYYYDDMM=6	The format to be used for the date in each audit record
Escape Character	A string Required	Any	The escape character to be used in audit records
RecordFormat	A string Required	Any	The format for the audit record
EventMapping	An array of strings associated with event types	Any	The mapping of audit events to character strings to be used in the audit record

Java

```
public int getDateFormat();
public char getEscapeCharacter();
```

```
public String getRecordFormat();
public String getEventMapping(int eventType);
```

The get method used to work with event mapping in the Master Audit Rule is unusual. The array of EventMapping strings consists of a series of pairs of data. Each pair is the association of a character string with an event type. The event type is provided as an enumerated argument to the getEventMapping method to return the character string associated with it.

The enumerated values for `eventtype` are the following:

```
AUTHENTICATION_FAILURE = 0
AUTHENTICATION_SUCCESS = 1
AUTHORIZATION_FAILURE = 2
AUTHORIZATION_SUCCESS= 3
```

Class ObAMMasterAuditRuleMgd

(Managed Code) An ObAMMasterAuditRule object represents the master audit rule, which specifies global audit parameters and defaults to be used if there is no audit rule specified for a specific policy.

Label	Type	Range	Description
DateFormat	Enum Required	An integer from the following enumerated set of possible date formats: UNDEFINED=0 INTEGER = 1 MMDDYYYY=2 DDMMYYYY=3 ISO8601 = 4 YYYYMMDD=5 YYYYDDMM=6	The format to be used for the date in each audit record
Escape Character	A string Required	Any	The escape character to be used in audit records
RecordFormat	A string Required	Any	The format for the audit record
EventMapping	An array of strings associated with event types	Any	The mapping of audit events to character strings to be used in the audit record

Managed Code Form

```
// getters
```

```
System::String *getEventMapping(ObAMAuditRule_EventTypeMgd
    *eventType);
__property ObAMMasterAuditRule_DateFormat get_DateFormat();
__property const char get_EscapeCharacter();
__property System::String *get_RecordFormat();
```

Access Policy Objects

Access policy objects represent certain basic object types for which information can be created, modified, deleted, and read through the Access Manager GUI.

From the perspective of a policy domain, there are several tiers of objects in which the higher-level objects include objects subordinate to them. This perspective is not a true one in that policies which a policy domain contains can include their own rules and expressions containing access conditions and actions. However, taking policies into account, from a hierarchical perspective, the objects are:

Level 1—"Class ObAMPolicyDomain" on page 446

Level 2—"Class ObAMAdminRule" on page 452

Level 3—"Class ObAMAuthorizationExpr" on page 471.

Level 4—"Class ObAMAuthorizationRule" on page 464

Level 5—"Class ObAMAuthenticationRule" on page 460 and "Class ObAMAuthorizationRule" on page 464

Level 6—"Class ObAMAccessConditions" on page 478 and "Class ObAMTimingConditions" on page 483

Level 7—"Class ObAMAction" on page 488, "Class ObAMAuditRule" on page 490, "Class ObAMDate" on page 492, "Class ObAMIdentity" on page 494, "Class ObAMParameter" on page 496, "Class ObAMTime" on page 498

Level 8—"Class ObAMPolicy" on page 454

Description of these classes follows the form that is used for the "Access System Configuration Objects" on page 434.

Definitions for the enumerated values and methods of these classes are given in "Access Management API Definitions" on page 621.

About String Names

Many objects contain a name data member whose value you specify as a string, and a method to set that value, as shown in the following method definition:

```
public void setName(String value);
```

To refer to the object elsewhere—for example, passing it as a parameter to a method of another class—you must specify exactly the string that you gave for that name when you created the object.

For example, if you created an authorization rule called "Authz Rule 1" as is done in the following snippet of code,

```
ObAMAuthorizationRule authzRule1 = new
ObAMAuthorizationRule();
authzRule1.setName("Authz Rule 1");
authzRule1.setEnabled(true);
```

and then you created another authorization rule called "Authz Rule 2", you could use those rules in an authorization expression.

To create an authorization expression containing the two rules, you would refer to each one of them by the exact string given as its name, as shown in the following line of code:

```
p1_authzExpr.setExpression(
"Authz Rule 1 & Authz Rule 2");
```

This concept applies to all objects having a name data member.

Class ObAMPolicyDomain

An object of the ObAMPolicyDomain class represents a NetPoint access policy domain that includes a set of resources and the authentication rule and authorization expression that control access to those resources. Optionally, a policy domain can contain audit rules. A policy domain also specifies the administrators who can manage the domain. For details about policy domains and protecting reources with policy domains, see the *NetPoint 7.0 Administration Guide Volume 2*.

A policy domain can also include one or more policies, each of which can have its own rules and authorization expression. For information, see "Class ObAMPolicy" on page 454.

Adding Objects to a Policy Domain—When you add to a policy domain a previously created object containing a named data member, you must refer to the object by the exact string given as the name when you created the object. For details, see "About String Names" on page 445.

Label	Type	Range	Description
Name (key)	A string Required	Any	A unique name
Description	A string Optional	Any	The displayed description for the Policy Domain

Label	Type	Range	Description
Enabled	A Boolean flag, Optional	0, meaning no, or other, meaning yes.	A flag indicating if the Policy Domain is enabled
Resource	An array of resource objects Optional	See the class definition on page 497.	A set of resources with resource types, host IDs, and URL prefixes that identify the world to which the policy domain applies
Default Authentication Rule	One Authentication Rule object Optional	See the class definition on page 460.	An authentication rule that specifies, among other things, the challenge method to be used to obtain credentials from the user.
Default Authorization Expression	One Authorization Expression object per policy domain	See the class definition on page 471.	An authorization expression contains one or more authorization rules whose collective evaluation determines if the requesting user is granted access to the resource The authorization rules of an expression must be specified as strings matching exactly the names of the rules given when they were created. For details, see “Class ObAMAuthorizationRule” on page 464 and “Class ObAMAuthorizationExpr” on page 471. Rules of an authorization expression are evaluated based on precedence and priority. See details about vvaluation of the rules of an expression in the *NetPoint 7.0 Administration Guide Volume 2*.
Authorization Rule	One or more Authorization Rule objects per policy domain	See the class definition on page 464.	A rule specifying who is allowed or not allowed to use a protected resource and under what conditions. Authorization Rules are included in Authorization Expressions. To specify an authorization rule in an authorization expression, the string given as its name must be specified exactly. Authorization Rules created for a policy domain can be used for the policy domain or any of its policies.

Label	Type	Range	Description
Default AuditRule	One Audit Rule object Optional	See the class definition on page 490	An audit rule that is used if no policy-specific audit rule applies
Policy	An array of Policy objects Optional	See the class definition on page 454	A policy that further qualifies access requirements for the set of resources it applies to.
Delegate AdminRule	One AdminRule object Optional	See the class definition on page 452	An admin rule that specifies who is allowed to delegate administration rights for this policy domain, including the right to further delegate rights
Grant AdminRule	One AdminRule object Optional	See the class definition on page 452	An admin rule that specifies who is allowed to grant basic rights for this Policy Domain to other users
Basic AdminRule	One AdminRule object Optional	See the class definition on page 452	An admin rule that specifies who is allowed basic rights to manage this Policy Domain

Java

```
public String getName();
public String getDescription();
public boolean getEnabled();
public int getEnabled();
public int getNumberOfResources();
public ObAMResource getResource(int index);
public ObAMAuthenticationRule
        getDefaultAuthenticationRule(int index);
public int getNumberOfPolicies();
public OAMPolicy getPolicy(int index);
public int getNumberOfAuthorizationRules();
public ObAMAuthorizationRule getAuthorizationRule(int index);
public ObAMAuthorizationExpr getDefaultAuthorizationExpr();
public ObAMAuditRule getDefaultAuditRule();
public ObAMAdminRule getDelegateAdminRule();
public ObAMAdminRule getGrantAdminRule();
public ObAMAdminRule getBasicAdminRule();
public void modifyAuthorizationRule(
        ObAMAuthorizationRule value);
public void modifyPolicy(ObAMPolicy value);
public void modifyResource(ObAMResource value);
public void setDescription(String value);
public void setName(String value);
public void setEnabled(boolean value);
```

```
public void setDefaultAuthenticationRule(
        ObAMAuthenticationRule value);
public void setDefaultAuthorizationExpr(
        ObAMAuthorizationExpr value);
public void setDefaultAuditRule(ObAMAuditRule value);
public void setDelegateAdminRule(ObAMAdminRule value);
public void setGrantAdminRule(ObAMAdminRule value);
public void setIDFrom(ObAMPolicyDomain other);
public void setBasicAdminRule(ObAMAdminRule value);
public void addResource(ObAMResource value);
public void addAuthorizationRule(ObAMAuthorizationRule value);
public void addPolicy(ObAMPolicy value);
public void removeResource(ObAMResource value);
public void removeAuthorizationRule(
        ObAMAuthorizationRule value);
public void removePolicy(ObAMPolicy value);
```

Obsoleted Methods

Do not use the following methods. They have been obsoleted as of NetPoint Access Manager 6.5:

```
getNumberOfDefaultAuthorizationRules();
getDefaultAuthorizationRule();
addDefaultAuthorizationRule();
removeDefaultAuthorizationRule();
```

Creating a Policy Domain

Listing 35 shows an excerpt from an example program that creates a policy domain called My Domain to protect certain resources, and then it enables the domain.

The code that would call the rest of the methods necessary to define a policy domain—methods to set the default authentication rule, add authorization rules to the policy domain, set the default authorization expression, and so forth—is not shown here. Those ObAMPolicyDomain methods and others add objects to the policy domain, and they are called only after the objects are created.

The code excerpt in Listing 35 performs the following functions:

- It creates a new policy domain object (domain), and it sets the name of the policy domain to My Domain.

  ```
  ObAMPolicyDomain domain = new ObAMPolicyDomain();
  domain.setName("My Domain");
  ```

- It sets a description for the policy domain

  ```
  domain.setDescription("This domain was created by the
          ObAM example program.");
  ```

- It enables the policy domain. (You must enable a policy domain before you can use it.)

```
domain.setEnabled(true);
```

- It creates an ObAMResource object called resource and sets the resource type to http.

```
ObAMResource resource = new ObAMResource();
resource.setResourceType("http");
```

- Using the new resource object, it specifies the location of the resources to be included in the policy domain. It sets the hostID and it sets the URL prefix.

  Together with the hostID, the URL defines the path for the resources protected by the policy domain (http://host1/myresources).

  All resources added to a policy domain are identified by the hostID for the host on which they reside and their URLs.

```
resource.setHostID("host1");
resource.setURLPrefix("/myresources");
```

- It adds the resources to the policy domain

```
domain.addResource(resource);
```

Listing 35 Creating a Policy Domain

```
                .
                .
                .
ObAMPolicyDomain domain = new ObAMPolicyDomain();
domain.setName("My Domain");
domain.setDescription("This domain was created by the ObAM example
                       program.");
domain.setEnabled(true);
ObAMResource resource = new ObAMResource();
resource.setResourceType("http");
resource.setHostID("host1");
resource.setURLPrefix("/myresources");
domain.addResource(resource);
                .
                .
                .
```

Class ObAMPolicyDomainMgd

(Managed Code) An object of the ObAMPolicyDomainMgd class represents a NetPoint access policy domain that determines if access to a set of resources is authorized for a set of users meeting certain conditions. A Policy Domain contains default rules and it can contain policies that further qualify access requirements for subsets of resources. A policy domain also specifies the administrators who can manage the domain. For details on the object class components, see “Class ObAMPolicyDomain” on page 446.

Managed Code Form

```
// Getters and setters
__property System::String *get_Name();
__property System::String *get_Description();
__property bool get_Enabled();
__property int get_NumberOfResources();
__property int get_NumberOfAuthorizationRules();
__property int get_NumberOfPolicies();
ObAMResourceMgd *getResource(int index);
ObAMAuthorizationRuleMgd *getAuthorizationRule(int index);
__property ObAMAuthorizationExprMgd
               *get_DefaultAuthorizationExpr();
ObAMPolicyMgd *getPolicy(int index);
__property ObAMAuthenticationRuleMgd
               *get_DefaultAuthenticationRule();
__property ObAMAuditRuleMgd *get_DefaultAuditRule();
__property ObAMAdminRuleMgd *get_DelegateAdminRule();
__property ObAMAdminRuleMgd *get_GrantAdminRule();
__property ObAMAdminRuleMgd *get_BasicAdminRule();
__property void set_IDFrom(ObAMPolicyDomainMgd *other);
__property void set_Name(System::String *value);
__property void set_Description(System::String *value);
__property void set_Enabled(bool value);
__property void
   set_DefaultAuthenticationRule(ObAMAuthenticationRuleMgd *value);
__property void
   set_DefaultAuthorizationExpr(ObAMAuthorizationExprMgd *expr);
__property void set_DefaultAuditRule(ObAMAuditRuleMgd *value);
__property void set_DelegateAdminRule(ObAMAdminRuleMgd *value);
__property void set_GrantAdminRule(ObAMAdminRuleMgd *value);
__property void set_BasicAdminRule(ObAMAdminRuleMgd *value);
__property void set_AddResource(ObAMResourceMgd *value);
__property void
   set_AddAuthorizationRule(ObAMAuthorizationRuleMgd *value);
__property void set_AddPolicy(ObAMPolicyMgd *value);
__property void set_ModifyResource(ObAMResourceMgd *value);
__property void set_ModifyPolicy(ObAMPolicyMgd *value);
__property void
   set_ModifyAuthorizationRule(ObAMAuthorizationRuleMgd *value);
__property void set_RemoveResource(ObAMResourceMgd *value);
__property void
```

```
   set_RemoveAuthorizationRule(ObAMAuthorizationRuleMgd *value);
__property void set_RemovePolicy(ObAMPolicyMgd *value);
```

Class ObAMAdminRule

An object of the ObAMAdminRule class specifies users who are authorized to administer a policy domain.

Label	Type	Range	Description

Role
(no key)
	An array of strings.
	At least one required.
		One of:
		`none`
		`anyone`
		`noone`
			A set of roles.
			
			NOTE: The array format here allows for future use. For NetPoint 6, the “array” *is limited to one value*.

Person	An array of Identity objects. Optional, but at least one of `Person`, `Group`, or `Rule` is required.	See the class definition on page 494.	A set of people.
Group	An array of Identity objects. Optional, but at least one of `Person`, `Group`, or `Rule` is required.	See the class definition on page 494.	A set of named groups (of people).
Rule	An array of strings. Optional, but at least one of `Person`, `Group`, or `Rule` is required.	Any	A set of LDAP rules that select user profiles.

Java

```
public int getNumberOfPersons();
public int getNumberOfGroups();
public int getNumberOfRoles();
public int getNumberOfRules();
public ObAMIdentity getPerson(int index);
public ObAMIdentity getGroup(int index);
public String getRole(int index);
public String getRule(int index);
public void addRole(String value);
public void addPerson(ObAMIdentity value);
```

```
public void addGroup(ObAMIdentity value);
public void addRule(String value)
public void removeRole(String value);
public void removePerson(ObAMIdentity value);
public void removeGroup(ObAMIdentity value);
public void removeRule(String value);
public void setIDFrom(ObAMAdminRule other);
```

Creating an Administrator Rule

The code in Listing 36 performs the following functions:

- It creates an administrator rule object (ObAMAdminRule) to be used to assign to someone Delegated Administrator rights and responsibilities for the policy domain called My Domain.

  ```
  ObAMAdminRule adminRule = new ObAMAdminRule();
  ```

  This listing presupposes that the policy domain was created previously and called My Domain.

- It creates an identity object (ObAMIdentity) called adminPerson to be used to identify the person to be assigned administrative rights for the domain.

  ```
  ObAMIdentity adminPerson = new ObAMIdentity();
  ```

- Using the methods of the ObAMIdentity class, it sets the login ID for the person designated administrator of the policy domain.

  ```
  adminPerson.setLoginID("A.Loomis");
  ```

- Then it adds the ObAMIdentity object called adminPerson to the administrator rule created at the outset of the code excerpt.

  ```
  adminRule.addPerson(adminPerson);
  ```

- Finally, it calls the ObAMPolicyDomain's setDelegateAdminRule method to add the adminRule rule to the policy domain called My Domain.

  ```
  domain.setDelegateAdminRule(adminRule);
  ```

Listing 36 Assigning Administrator Rights

```
                    .
                    .
                    .
ObAMAdminRule adminRule = new ObAMAdminRule();
ObAMIdentity adminPerson = new ObAMIdentity();
adminPerson.setLoginID("A.Loomis");
adminRule.addPerson(adminPerson);
domain.setDelegateAdminRule(adminRule);
                    .
                    .
                    .
```

Class ObAMAdminRuleMgd

(Managed Code). An object of the ObAMAdminRule class specifies users who are authorized to administer a policy domain. For details on the object class components, see “Class ObAMAdminRule” on page 452.

Managed Code Form

```
// Getters and setters
__property int get_NumberOfPersons();
__property int get_NumberOfGroups();
__property int get_NumberOfRoles();
__property int get_NumberOfRules();
ObAMIdentityMgd *getPerson(int index);
ObAMIdentityMgd *getGroup(int index);
System::String *getRole(int index);
System::String *getRule(int index);
__property void set_IDFrom(ObAMAdminRuleMgd *other);
__property void set_AddRole(System::String *value);
__property void set_AddPerson(ObAMIdentityMgd *person);
__property void set_AddGroup(ObAMIdentityMgd *group);
__property void set_AddRule(System::String *value);
__property void set_RemoveRole(System::String *value);
__property void set_RemovePerson(ObAMIdentityMgd *person);
__property void set_RemoveGroup(ObAMIdentityMgd *group);
__property void set_RemoveRule(System::String *value);
```

Class ObAMPolicy

An object of the ObAMPolicy class represents a NetPoint access policy within a policy domain. The policy determines who can access a set of resources within the policy domain protected by the policy. It specifies the users allowed or denied access and the conditions controlling access. If any of the Authentication or Audit rules or the Authorization Expression in the policy are omitted, the corresponding default rule or expression for the policy domain that includes the policy is used. For background information about policies nd policy domains, see the *NetPoint 7.0 Administration Guide Volume 2*.

Adding Objects to a Policy—When you add to a policy a previously created object containing a named data member, you must refer to the object by the exact string given as the name when you created the object. For example, you must specify the names of authorization rules as strings matching exactly the names that were given when the authorization rules were created and added to the policy domain. For details, see “About String Names” on page 445.

Label	Type	Range	Description
Name (key)	A string Required	Any	A unique name

Label	Type	Range	Description
Description	A string Optional	Any	A description
ResourceType	A string Optional	Any	The resource type for the set of resources to which the policy applies
Operations	An array of strings Required	Any	A set of access operations (defined for the resource type) to which the policy applies.
HostID	A string Optional	Any	A host ID for the set of resources.
Resources	An array of Resource objects Required	See the class definition on page 497.	A set of resources that further qualify the set of resources.
URLPattern	A string Optional	Any	A URL pattern that further qualifies the set of resources.
QueryString	A string Optional	Any	A query string to be matched against request data.
Parameters	A array of Parameter objects Optional	See the class definition on page 496.	A set of parameters to be matched against request data.
AuthenticationRule	One Authentication Rule object Optional	See the class definition on page 460.	An authentication rule includes an authentication scheme, which among other things, specifies the challenge method used to obtain credentials from the user. You must specify an authentication scheme that has already been created.

Label	Type	Range	Description
Authorization Expression	One Authorization Expression per policy Optional	See the class definition on page 471.	An authorization expression contains one or more authorization rules whose evaluation determines if the user requesting the resource is granted access to it. The authorization rules of an expression must be specified as strings matching exactly the names of the rules given when they were created. If a policy does not include an authorization expression, the expression for the policy domain applies. For details, see “Class ObAMAuthorizationRule” on page 464 and “Class ObAMAuthorizationExpr” on page 471. Rules of an authorization expression are evaluated based on precedence and priority. See details about evaluation of the rules of an expression and configuring user authorization in the *NetPoint 7.0 Administration Guide Volume 2*.
AuditRule	One AuditRule object Optional	See the class definition on page 490.	An audit rule that specifies how the access is to be audited.

Java

```
public String getName();
public String getDescription();
public String getResourceType();
public String getHostID();
public String getURLPattern();
public String getQueryString();
public int getNumberOfOperations();
public int getNumberOfResources();
public int getNumberOfParameters();
public ObAMAuthorizationExpr getAuthorizationExpr();
public ObAMAuthenticationRule getAuthenticationRule();
public String getOperation(int index);
public ObAMResource getResource(int index);
public ObAMParameter getParameter(int index);
public ObAMAuditRule getAuditRule();
```

```
                public void setName(String value);
                public void setDescription(String value);
                public void setResourceType(String value);
                public void setHostID(String value);
                public void setURLPattern(String value);
                public void setQueryString(String value);
                public void setAuthorizationExpr(ObAMAuthorizationExpr
value);
                public void setAuthenticationRule(ObAMAuthenticationRule
value);
                public void setAuditRule(ObAMAuditRule value);
                public void setIDFrom(ObAMPolicy other);
                public void addOperation(String value);
                public void addResource(ObAMResource value);
                public void addParameter(ObAMParameter value);
                public void removeOperation(String value);
                public void removeResource(ObAMResource value);
                public void removeParameter(ObAMParameter value);
                public void modifyParameter(ObAMParameter value);
                public void modifyResource(ObAMResource value);
```

Obsoleted Methods

Do not use the following methods. They have been obsoleted as of the NetPoint 6.5 Access Manager:

```
getNumberOfAuthorizationRules();
getAuthorizationRule();
addAuthorizationRule();
removeAuthorizationRule();
```

Creating a Policy

The code in Listing 37 creates a policy called My Domain Policy 1. The policy protects a resource within the policy domain called My Domain. My Domain was created previously by the application this code was excerpted from.

The code in this listing sets an authorization expression for the policy domain. An authorization expression includes authorization rules, which together are used to control access to resources of the policy domain.

The code performs the following functions:

- It creates a policy object called policy1 for the new policy, and it sets the name of the policy to My Domain Policy 1.

```
ObAMPolicy policy1 = new ObAMPolicy();
policy1.setName("My Domain Policy 1");
```

- It specifies the type of resource the policy applies to, and it identifies the resource by giving its host ID and URL.

```
policy1.setResourceType("http");
```

```
policy1.setHostID("host1");
policy1.setURLPattern("/myresources/doc1.html");
```

- It defines for the My Domain Policy 1 policy the kinds of operations that can be performed on the protected http resource. The GET and POST operations are allowed.

```
policy1.addOperation("GET");
policy1.addOperation("POST");
```

- It adds the resource to the My Domain Policy 1 policy.

```
policy1.addResource(resource);
```

- It creates an ObAMAuthorizationExpr object called p1_authzExpr to be used for the My Domain Policy 1 policy's authorization protection.

```
ObAMAuthorizationExpr p1_authzExpr = new;
ObAMAuthorizationExpr();
```

- It defines the authorization expression for the p1_authzExpr object. The expression consists of authorization rule 1 (Authz Rule 1) and authorization rule 2 (Authz Rule 2). These are the names of the rules given as unique strings when the rules were created. To identify the rules of an expression, you must specify exactly the strings given as their names. For details, see "Class ObAMAuthorizationRule" on page 464 and "Class ObAMAuthorizationExpr" on page 471.

```
p1_authzExpr.setExpression(
"Authz Rule 1 & Authz Rule 2");
```

  For details about how authorization expressions are evaluated see details about authorization expression evaluation and configuring user authorization in the *NetPoint 7.0 Administration Guide Volume 2*.

- It sets the duplicate actions policy for the p1_authzExpr authorization expression, and then it adds the authorization expression to the My Domain Policy 1 policy.

```
p1_authzExpr.setDuplicateActionsPolicy(
        ObAMAuthorizationExpr.UNDEFINED);
policy1.setAuthorizationExpr(p1_authzExpr);
```

  The duplicate actions policy for My Domain Policy 1 takes precedence over the one set for the default authorization expression for the policy domain My Domain.

  For information explaining what duplicate actions are and how they are handled for a policy based on your specification, see "Class ObAMAuthorizationExpr" on page 471. also, see details about dDuplicate actions and configuring user authorization in the *NetPoint 7.0 Administration Guide Volume 2*.

- Finally, it adds the My Domain Policy 1 policy to the My Domain domain.

```
domain.addPolicy(policy1);
```

Listing 37 Creating Policies Within the My Domain Policy Domain

```
.
.
.

ObAMPolicy policy1 = new ObAMPolicy();
policy1.setName("My Domain Policy 1");
policy1.setResourceType("http");
policy1.setHostID("host1");
policy1.setURLPattern("/myresources/doc1.html");
policy1.addOperation("GET");
policy1.addOperation("POST");
policy1.addResource(resource);
ObAMAuthorizationExpr p1_authzExpr = new ObAMAuthorizationExpr();
p1_authzExpr.setExpression("Authz Rule 1 & Authz Rule 2");
p1_authzExpr.setDuplicateActionsPolicy(ObAMAuthorizationExpr.UNDEFINED);
policy1.setAuthorizationExpr(p1_authzExpr);
domain.addPolicy(policy1);
```

Class ObAMPolicyMgd

(Managed Code). An ObAMPolicyMgd object represents a NetPoint access policy that determines if access to a set of resources is authorized for a set of users meeting certain conditions. If any of the Authentication or Audit rules or the Authorization Expression are omitted, the corresponding rule in the policy domain that contains the policy will be used. For details on the object class components, see “Class ObAMPolicy” on page 454.

Managed Code Form

```
// Getters and setters
__property System::String *get_Name();
__property System::String *get_Description();
__property System::String *get_ResourceType();
__property System::String *get_HostID();
__property System::String *get_URLPattern();
__property System::String *get_QueryString();
__property int get_NumberOfOperations();
__property int get_NumberOfResources();
__property int get_NumberOfParameters();
System::String *getOperation(int index);
ObAMResourceMgd *getResource(int index);
ObAMParameterMgd *getParameter(int index);
__property ObAMAuthorizationExprMgd *get_AuthorizationExpr();
__property ObAMAuthenticationRuleMgd *get_AuthenticationRule();
__property ObAMAuditRuleMgd *get_AuditRule();
__property void set_IDFrom(ObAMPolicyMgd *other);
__property void set_Name(System::String *value);
__property void set_Description(System::String *value);
```

```
__property void set_ResourceType(System::String *value);
__property void set_HostID(System::String *value);
__property void set_URLPattern(System::String *value);
__property void set_QueryString(System::String *value);
__property void set_AuthenticationRule(
        ObAMAuthenticationRuleMgd *rule);
__property void set_AuthorizationExpr(
        ObAMAuthorizationExprMgd *expr);
__property void set_AuditRule(ObAMAuditRuleMgd *rule);
__property void set_AddOperation(System::String *value);
__property void set_AddResource(ObAMResourceMgd *resource);
__property void set_AddParameter(ObAMParameterMgd *parameter);
__property void set_ModifyResource(ObAMResourceMgd *resource);
__property void set_RemoveOperation(System::String *value);
__property void set_RemoveResource(ObAMResourceMgd *resource);
__property void set_RemoveParameter(ObAMParameterMgd *parameter);
```

Class ObAMAuthenticationRule

An object of the ObAMAuthenticationRule class specifies how authentication is to be performed when users request access to resources protected by the rule. Every policy domain must include one and only one default authentication rule containing an ObAMAuthenticationScheme object. Optionally, each policy a policy domain contains can include its own authentication rule. If it does not include one, the policy is protected by the policy domain's authentication rule.

Authentication Schemes—An authentication rule must contain an authentication scheme, which specifies, among other information, the challenge method used to obtain the user's credentials and authenticate the user. You can use a NetPoint-provided authentication scheme, or you can use a custom one. In any case, you must specify an authentication scheme that has already been created. To get a list of existing authentication schemes, you must use the Access System Console. See details about configuring user authentication in the *NetPoint 7.0 Administration Guide Volume 2*.

Actions for Authentication Rule Objects—For an authentication rule object, you can set the kinds of actions to be taken if authentication of the user is successful. You can also set the kinds of actions to be taken if user authentication fails.

When you create it, the ObAMAuthenticationRule class inherits the ObAMObjectWithActions class. It uses the methods of this class to manipulate any information pertaining to actions for the authentication rule. Do not instantiate directly the ObAMObjectWithActions class because the ObAMAuthenticationRule class inherits it automatically. For details, see "Class ObAMObjectWithActions" on page 480.

For conceptual information about authentication rules and schemes, including the kinds of challenge methods that can be used to authenticate users, see the *NetPoint 7.0 Administration Guide Volume 2*.

Label	Type	Range	Description
Name (key)	A string Required	Any	A unique name.
Description	A string Optional	Any	A description.
Authentication Scheme	A string Required	Any	An authentication scheme that specifies how credentials are to be obtained and processed.
Action Type	An enumerated value, integer Required	SUCCESS = 0 FAILURE = 1 INCONCLUSIVE = 2 See the class definition on page 488.	A value that identifies the type of action that the method applies to. For actions, the ObAMAuthenticationRule uses the methods of the ObAMObjectWithActions class, which it inherits.

Java

```
public ObAMAuthenticationRule();
public String getDescription();
public String getScheme();
public void setDescription(String value);
public void setScheme(String value);
public void setIDFrom(ObAMObjectWithActions other);
public int getNumberOfActions(
                        int actionType) throws ObAMException;
public ObAMAction getActionOfType(int actionType, int index)
                        throws ObAMException;
public void addActionOfType(int actionTtype, ObAMAction value)
                        throws ObAMException;
public void removeActionOfType(int actionType, ObAMAction value)
                        throws ObAMException;
public String getName();
public void setName(String value);
```

C Form

```
typedef const void * ObAMAuthenticationRule_t;
ObAMAuthenticationRule_t ObAMAuthenticationRule_new();
ObAMAuthenticationRule_t ObAMAuthenticationRule_copy(
                        ObAMAuthenticationRule_t authn);
void ObAMAuthenticationRule_delete(
```

```
                        ObAMAuthenticationRule_t *pAuthn);
const char *ObAMAuthenticationRule_getName(
                        ObAMAuthenticationRule_t authn);
const char *ObAMAuthenticationRule_getDescription(
                        ObAMAuthenticationRule_t authn);
const char *ObAMAuthenticationRule_getScheme(
                        ObAMAuthenticationRule_t authn);
void ObAMAuthenticationRule_setIDFrom(
                        ObAMAuthenticationRule_t authn,
                        ObAMAuthenticationRule_t other);
void ObAMAuthenticationRule_setName(
                        ObAMAuthenticationRule_t authn,
                        const char *value);
void ObAMAuthenticationRule_setDescription(
                        ObAMAuthenticationRule_t authn,
                        const char *value);
void ObAMAuthenticationRule_setScheme(
                        ObAMAuthenticationRule_t authn,
                        const char *value);
int ObAMAuthenticationRule_getNumberOfActions(
                      ObAMObjectWithActions_ActionType type,
                        ObAMAuthenticationRule_t authn);
ObAMAction_t ObAMAuthenticationRule_getActionOfType(
                      ObAMObjectWithActions_ActionType type,
ObAMAuthenticationRule_t authn, OBAMAction_t value);
                      ObAMAuthenticationRule_t authn, int index);
void ObAMAuthenticationRule_removeActionOfType(
                      ObAMObjectWithActions_ActionType type,
                      ObAMAAutbenticationRule_t authn,
                      ObAMAction_t value);
```

Obsoleted Methods

Do not use the following methods. They have been obsoleted in the NetPoint 6.5 Access Manager:

```
ObAMAuthenticationRule_getNumberOfSuccessActions();
ObAMAuthenticationRule_getNumberOfFailureActions();
ObAMAction_t ObAMAuthenticationRule_getSuccessAction();
ObAMAction_t ObAMAuthenticationRule_getFailureAction();
ObAMAuthenticationRule_addSuccessAction();
ObAMAuthenticationRule_addFailureAction();
ObAMAuthenticationRule_removeSuccessAction();
ObAMAuthenticationRule_removeFailureAction();
```

Creating an Authentication Rule

The code in Listing 38 creates an authentication rule for the My Domain policy domain object. The code sets the default authentication rule for My Domain to use the Basic over LDAP scheme.

The code performs the following functions:

- It creates an ObAMAuthenticationRule object called authnRule to be used for the default authentication rule, and it sets the name of the rule to authnRule.

```
ObAMAuthenticationRule authnRule = new;
ObAMAuthenticationRule();
authnRule.setName("My Domain Default Authn Rule");
```

- It sets the scheme to be used for the default authentication rule to NetPoint Basic Over LDAP.

```
authnRule.setScheme(“NetPoint Basic Over LDAP”);
```

- It defines an action to be performed if authentication of the user is successful, and it adds the action to the default authentication rule.

```
authnAction2.setType("otherType");
authnAction2.setName("authnAction");
authnAction2.setValue("z");
authnAction2.setValueType(ObAMAction.FIXEDVALUE);
authnRule.addSuccessAction(authnAction2);
```

- It adds the default authentication rule to the My Domain policy domain created previously by the application from which this code is excerpted.

```
domain.setDefaultAuthenticationRule(authnRule);
```

Listing 38 Creating a Default Authentication Rule for a Policy Domain

```
                                .
                                .
                                .
ObAMAuthenticationRule authnRule = new ObAMAuthenticationRule();
authnRule.setName("My Domain Default Authn Rule");
authnRule.setScheme(“NetPoint Basic Over LDAP”);
ObAMAction authnAction2 = new ObAMAction();
authnAction2.setType("otherType");
authnAction2.setName("authnAction");
authnAction2.setValue("z");
authnAction2.setValueType(ObAMAction.FIXEDVALUE);
authnRule.addSuccessAction(authnAction2);
domain.setDefaultAuthenticationRule(authnRule);
                                .
                                .
                                .
```

Class ObAMAuthenticationRuleMgd

(Managed Code). An ObAMAuthenticationRule object specifies how authentication is to be performed for access to resources covered by a policy or policy domain. For details on object class components, see “Class ObAMAuthenticationRule” on page 460.

Managed Code

```
// Getters and setters
__property System::String *get_Name();
__property System::String *get_Description();
__property System::String *get_Scheme();
int getNumberOfActions(ObAMActionTypeMgd *action);
ObAMActionMgd *getActionOfType(ObAMActionTypeMgd *type, int index);
__property void set_IDFrom(ObAMAuthenticationRuleMgd *other);
__property void set_Name(System::String *value);
__property void set_Description(System::String *value);
__property void set_Scheme(System::String *value);
void addActionOfType(ObAMActionTypeMgd *action,
                        ObAMActionMgd *value);
void modifyActionOfType(ObAMActionTypeMgd *action,
                        ObAMActionMgd *value);
void removeActionOfType(ObAMActionTypeMgd *action,
                        ObAMActionMgd *value);
```

Class ObAMAuthorizationRule

An object of the ObAMAuthorizationRule class specifies the conditions for allowing or denying user access to the resources it protects. An authorization rule contains an authorization scheme. It can also contain actions to be returned depending on the outcome of the attempt to authorize the user requesting access to the protected resource. Actions can be associated with a result of Success or Failure.

An authorization rule can:

- Appear in more than one authorization expression.
- Appear in a single authorization expression more than once.

Any of the authorization rules you create can be used in an authorization expression for a policy domain or any of its policies.

It is the result of evaluation of the expression—that is, all of the rules it contains and the way in which they are combined—that determines the access controls for the protected resources.

For conceptual details on authorization rules and their contents and authorization expression evaluation, see the *NetPoint 7.0 Administration Guide Volume 2*.

Authorization Schemes—An authorization rule must contain an authorization scheme. You can use the NetPoint default authorization scheme, or you can use a custom one, if any custom authorization schemes have been created. To get a list of existing authorization schemes, you must use the Access System Console. For details about authorization schemes for custom plug-ins and configuring user authorization, see *NetPoint 7.0 Administration Guide Volume 2*.

About the Names of Authorization Rules and Authorization Expressions—To name an authorization rule, you specify a unique string. You use this string later in an authorization expression for a policy domain and its policies to identify rules an authorization expression contains. (Authorization rules are included in an authorization expression.) For each authorization rule an authorization expression contains, you must specify exactly the string given as the name of the authorization rule. For details, see "About String Names" on page 445. For details, see "Class ObAMAuthorizationExpr" on page 471.)

Actions for Authorization Rules—For an authorization rule, you can specify the kinds of actions to be taken based on the result of evaluation of the rule. You can specify actions to be taken if authorization succeeds as a result of the rule or if it fails.

When you create the ObAMAuthorizationRule object, it inherits automatically the ObAMObjectWithActions class, and it uses the methods of the ObAMObjectWithActions class for any functions pertaining to actions. Do not instantiate directly the ObAMObjectWithActions class. For details, see "Class ObAMObjectWithActions" on page 480.

Not all rules contribute to the result of an authorization expression. Those rules that do participate in the outcome of evaluation of the expression are referred to as determining rules. If a rule is a determining rule, its resulting actions are taken after the expression is evaluated.

Note: In the first version of the Access Management API (NetPoint version 6), custom Authorization Plug-ins are not supported by the Access Manager engine. For that reason, custom Authorization rules using custom Authorization Plug-ins cannot be created.

Label	Type	Range	Description
Name (key)	A string Required	Any	A unique name. This is the name that you specify in an authorization expression to include the rule in the expression.
Description	A string Optional	Any	A description.

Label	Type	Range	Description
Enabled	A Boolean flag Required	0, meaning no, or other, meaning yes.	Answers the question: Is the rule enabled?
AllowTakes Precedence	A Boolean flag Required	0, meaning no, or other, meaning yes	Answers the question: Do the allow conditions take precedence over the deny conditions?
Timing Conditions	An array of Timing Condition objects Optional	See the class definition on page 483.	Timing conditions specifying when the rule is in effect.
Action Type	An enumerated value, integer Required	SUCCESS = 0 FAILURE = 1 INCONCLUSIVE = 2 See the class definition on page 488.	A value that identifies the type of action that the method applies to. For actions, the ObAmAuthorizationRule uses the methods of the ObAMObjectWithActions class, which it inherits.
AllowAccess Conditions	One Access Condition object Optional	See the class definition on page 478.	Conditions under which access is allowed.
DenyAccess Conditions	One Access Condition object Optional	See the class definition on page 478.	Conditions under which access is denied.
Authorization Scheme	A string Optional	Any	The name of a custom or Oblix Authorization Scheme. You must specify the name of an authorization scheme that has already been created. To get a list of authorization schemes, use the Access System Console GUI. If this is provided, then it is illegal to also enter timing and access conditions.
Scheme Parameter	An array of Parameter Objects Optional	See the class definition on page 496	Parameters to be used with a custom Authorization Scheme

Java

```
public obAMAuthorizationRule();
public String getName();
public String getDescription();
public boolean getEnabled();
public boolean getAllowTakesPrecedence();
public int getNumberOfActions(int actionType)
                throws ObAMException;
public ObAMAction getActionOfType(int actionType, int index)
                throws ObAMException;
public void addActionOftype(int actionType, obAMAction value)
                throws ObAMException;
public void removeActionOfType(int actionType, ObAMAction value)
                throws ObAMException;
public ObAMTimingConditions getTimingConditions();
public int getNumberOfSchemeParameters();
public ObAMParameter getSchemeParameter(int index);
public ObAMAccessConditions getAllowAccessConditions();
public String getAuthorizationScheme();
public ObAMAccessConditions getDenyAccessConditions();
public void setName(String value);
public void setDescription(String value);
public void setEnabled(boolean value);
public void setAllowTakesPrecedence(boolean value);
public void setTimingConditions(
                ObAMTimingConditions value);
public void setAllowAccessConditions(
                ObAMAccessConditions value);
public void setDenyAccessConditions(
                ObAMAccessConditions value);
public void setAuthorizationScheme(String value);
public void addSchemeParameter(ObAMParameter value);
public void removeSchemeParameter(ObAMParameter value);
public void modifySchemeParameter(ObAMParameter value);
public void setIDFrom(ObAMAuthorizationRule other);
```

C

```
typedef const void * ObAMAuthorizationRule_t;
ObAMAuthorizationRule_t ObAMAuthorizationRule_new()
ObAMAuthorizationRule_t ObAMAuthorizationRule_copy(
             ObAMAuthorizationRule_t authz);
void ObAMAuthorizationRule_delete(
             ObAMAuthorizationRule_t *pAuthz);
const char *ObAMAuthorizationRule_getName(
             ObAMAuthorizationRule_t authz);
const char *ObAMAuthorizationRule_getDescription(
             ObAMAuthorizationRule_t authz);
ObAMAuthorizationRule_getEnabled(
             ObAMAuthorizationRule_t authz);
int ObAMAuthorization_getAllowTakesPrecedence(
             ObAMAuthorizationRule_t authz);
```

```
ObAMTimingConditions_t ObAMAuthorizationRule_getTimingConditions(
                ObAMAuthorizationRule_t authz);
   int ObAMAuthorizationRule_getNumberOfActions(
                ObAMObjectWithActions_ActionType type,
                ObAMAuthorizationRule_t authz);
   ObAMAction_t ObAMAuthorizationRule_getActionOfType(
                ObAMObjectWithActions_ActionType type,
                ObAMAuthorizationRule_t authz,
                int index);
   ObAMAccessConditions_t
                ObAMAuthorizationRule_getAllowAccessConditions(
                ObAMAuthorizationRule_t authz);
   ObAMAccessConditions_t
                ObAMAuthorizationRule_getDenyAccessConditions(
                ObAMAuthorizationRule_t authz);
   const char *ObAMAuthorizationRule_getAuthorizationScheme(
                ObAMAuthorizationRule_t authz);
   int ObAMAuthorizationRule_getNumberOfSchemeParameters(
                ObAMAuthorizationRule_t authz);
   ObAMParameter_t ObAMAuthorizationRule_getSchemeParameter(
                ObAMAuthorizationRule_t authz, int index);
   void ObAMAuthorizationRule_setIDFrom(
                ObAMAuthorizationRule_t authz,
                ObAMAuthorizationRule_t other);
```

Obsoleted Methods

Do not use the following methods. They have been obsoleted in the NetPoint 6.5 Access Management API.

```
ObAMAuthorizationRule_getSuccessAction();
ObAMAuthorizationRule_getFailureAction();
ObAMAuthorizationRule_getNumberOfSuccessActions();
ObAMAuthorizationRule_getNumberOfFailureActions();
ObAMAuthorizationRule_addSuccessAction();
ObAMAuthorizationRule_addFailureAction();
ObAMAuthorizationRule_removeSuccessAction();
ObAMAuthorizationRule_removeFailureAction();
```

Creating Authorization Rules

The code in Listing 39 creates an authorization rule and adds it to the My Domain policy domain object. The rule can be used for My Domain's default authorization expression and it can be used for the authorization expressions of any of the policies included in the My Domain policy domain. An application can create many instances of authorization rule objects to be used for a policy domain and its policies.

The code performs the following functions:

- It creates an ObAMAuthorizationRule object called authzRule1. It gives the rule the name "Authz Rule 1", and then it enables the rule. In the following

code segment the rule is named “Authz Rule 1”. If you were to include this rule in an authorization expression, you would identify it by specifying the string “Authz Rule 1”. For details, see “Creating an Authorization Expression” on page 477.

```
ObAMAuthorizationRule authzRule1 = new
ObAMAuthorizationRule();
authzRule1.setName("Authz Rule 1");
authzRule1.setEnabled(true);
```

- It creates an ObAMAccessConditions object called access1 to be used for the rule’s access conditions—whether Allow Access or Deny Access is not defined at this point.

```
ObAMAccessConditions access1 = new ObAMAccessConditions();
```

- It identifies to whom the access conditions apply—two individuals and one group.

  To identify each of these entities, the code:

  - Creates an ObAMIdentity object to set the login ID for the person or group.
  - Adds the person or group to the access1 object (ObAccessObject) of the ObAMAuthorizationRule authorization rule. The code which does this for person1 is shown in the following snippet.

```
ObAMIdentity person1 = new ObAMIdentity();
person1.setLoginID("A.Loomis");
access1.addPerson(person1);
```

- Adds the access1 object to the Authz Rule 1 (authzRule1) authorization rule object to set its Allow Access conditions.

```
authzRule1.setAllowAccessConditions(access1);
```

- Adds the Authz Rule 1 rule to the My Domain policy domain.

```
domain.addAuthorizationRule(authzRule1);
```

Listing 39 Creating an Authorization Rule

```
ObAMAuthorizationRule authzRule1 = new ObAMAuthorizationRule();
authzRule1.setName("Authz Rule 1");
authzRule1.setEnabled(true);
ObAMAccessConditions access1 = new ObAMAccessConditions();

ObAMIdentity person1 = new ObAMIdentity();
person1.setLoginID("A.Loomis");
access1.addPerson(person1);

ObAMIdentity person2 = new ObAMIdentity();
person2.setLoginID("E.Lawrence");
access1.addPerson(person2);
```

```
ObAMIdentity group = new ObAMIdentity();
group.setName("group1");
access1.addGroup(group);
person1.setLoginID("admin");
access1.addPerson(person1);

authzRule1.setAllowAccessConditions(access1);
ObAMAction action1 = new ObAMAction();
action1.setType("headerVar");
action1.setName("UserIs");
action1.setValue("uid");
action1.setValueType(ObAMAction.ATTRIBUTE);
authzRule1.addActionOfType(ObAMObjectWithActions.SUCCESS, action1);

domain.addAuthorizationRule(authzRule1);
```

Class ObAMAuthorizationRuleMgd

(Managed Code). An ObAMAuthorizationRule specifies the conditions for allowing or denying access to resources covered by a policy or policy domain. For details on the object class components, see “Class ObAMAuthorizationRule” on page 464.

Note: In the first version of the Access Management API (NetPoint version 6), custom Authorization Plug-ins are not supported by the Access Manager engine. For that reason, custom Authorization rules using custom Authorization Plug-ins cannot be created.

Managed Code Form

```
// Getters and setters
__property System::String *get_Name();
__property System::String *get_Description();
__property bool get_Enabled();
__property bool get_AllowTakesPrecedence();
__property ObAMTimingConditionsMgd *get_TimingConditions();
int getNumberOfActions(ObAMActionTypeMgd *action, int index);
ObAMActionMgd *getActionOfType(ObAMActionTypeMgd *action, int index);
__property ObAMAccessConditionsMgd
   *get_AllowAccessConditons();
__property ObAMAccessConditionsMgd
   *get_DenyAccessConditons();
__property System::String *get_AuthorizationScheme();
__property int get_NumberOfSchemeParameters();
ObAMParameterMgd *getSchemeParameter(int index);
__property void set_IDFrom(ObAMAuthorizationRuleMgd *other);
__property void set_Name(System::String *value);
__property void set_Description(System::String *value);
__property void set_Enabled(bool value);
__property void set_AllowTakesPrecedence(bool value);
```

```
__property void
     set_TimingConditions(ObAMTimingConditionsMgd *value);
__property void
     set_AllowAccessConditions(ObAMAccessConditionsMgd *value);
__property void
     set_DenyAccessConditions(ObAMAccessConditionsMgd *value);
void addActionOfType(ObAMActionTypeMgd *action, ObAMActionMgd *value);
void modifyActionOfType(ObAMActionTypeMgd *action, ObAMActionMgd *value);
void removeActionOfType(ObAMActionTypeMgd *action, ObAMActionMgd *value);
```

Class ObAMAuthorizationExpr

An object of the ObAMAuthorizationExpr class specifies an authorization expression. An authorization expression can consist of one or more authorization rules, specifying a simple or complex condition whose evaluation determines whether a user is granted access to a resource protected by the expression. In addition to authorization rules, an authorization expression contains symbols representing different ways to combine, and thus evaluate, the rules of the expression. You include in an authorization expression the names of the authorization rules you want to use.

It is an authorization expression that is included in a policy domain or a policy. Every policy domain must include a single authorization expression. A policy can include an authorization expression, but it is not a requirement. If it does not, the default authorization expression created for the policy domain is used for the policies resources.

Rules of an authorization expression can be combined in various ways to express particular authorization requirements. It is the result of the Access Server's evaluation of the authorization expression that determines if a user is given access to the requested resource.

An authorization expression includes:

- The authorization rules controlling user access to resources. If the expression contains more than one rule, the + and | symbols are used to specify how the rules are to be interpreted. For details on the content of authorization rules and how to use the + and | symbols to create the logic of the expression, see the *NetPoint 7.0 Administration Guide Volume 2*.
- A policy for dealing with duplicate actions.
- Optional actions to be performed by the client if evaluation of the expression succeeds, if it fails, or if the result is inconclusive.

About the Symbols Used in an Authorization Expression—If an authorization expression contains more than one authorization rule, you must include in it the symbols that specify the way in which those rules are combined and are to be interpreted for users requesting access to the protected resources. These symbols include + (AND) and | (OR) and parenthesis. They are referred to as operators. In the Access Manager GUI, either AND and OR can be specified or the symbols + and | can be specified as operators. However, for the Access Management API, you must use the symbols. For further explanation about the symbols and modifying an authorization scheme, see the *NetPoint 7.0 Administration Guide Volume 2*.

How Authorization Expressions Are Interpreted—It is possible to create complex authorization expressions. For this reason, it is important to understand how the Access System interprets those expressions in regard to precedence of operators and position of rules within the expression. For details about evaluation of the rules of an expression and configuring user authorization, see the *NetPoint 7.0 Administration Guide Volume 2*.

About an Expression Result of Inconclusive—If an expression evaluates to a result of Inconclusive, the NetPoint Access System returns a major status code of Deny and a minor status code of Inconclusive. The minor status code of Inconclusive is available to NetPoint 6.5 and later systems to allow those systems to distinguish between true Deny results and Deny results returned because of an Inconclusive state.

An authorization expression result of Deny differs from an authorization expression result of Inconclusive even though the user is denied access to the requested resource in both cases. Making this distinction gives you, as a developer, more options. For example, an application written to run with NetPoint 6.5 and later can interpret the two status codes for an Inconclusive result and use the additional information for other purposes. The application might then invoke other authorization engines instead of denying the user access to the resource.

Actions for an Authorization Expression—An authorization expression can have associated with it actions to be taken based on the result of evaluation of the expression. When it is created, the ObAMAuthorizationExpr objects inherits automatically the ObAMObjectWithActions class—do not directly instantiate this class.

The ObAMAuthorizationExpr object uses the methods of the ObAMObjectWithActions class to manipulate actions associated with it. These actions include

- Success Actions
- Failure Actions

- Inconclusive Actions

  An authorization expression is evaluated to a result of Inconclusive if the rules of the expression produce conflicting results. In this case the user is denied access to the resource. However, your application can use this information.

About the Result of an Authorization Expression and Actions Returned—It is important to understand which actions are returned after evaluation of an authorization expression. The actions of only those rules that contributed to the result of evaluation of the expression are returned. These rules are referred to as definitive rules. Because the concept of definitive rules is complex, you should review the explanation and examples in discussions on authorization rule evaluation and configuring user authorization in the *NetPoint 7.0 Administration Guide Volume 2*.

Duplicate Actions Policy—Because an authorization rule can be reused within an authorization expression, it is possible that evaluation of each instance of the authorization rule producing the same result can cause the Access Server to return the same action more than once. It is also possible that different rules of an expression could return the same actions. Conflict can occur when, as a result of evaluation of the expression, two or more rules contributing to the definitive result produce the same actions. You can set the policy for how duplicate actions are to be handled, if any occur. For this purpose, you use the following values

- ACTION_DUPLICATE = 0
- ACTION_IGNORE = 1
- ACTION_OVERWRITE = 2
- UNDEFINED = 3

Here is how these values are interpreted:

- **ACTION_DUPLICATE—**If you choose this option, the Access Server appends each new value it encounters to the information it returns to the application requesting authorization for the user. (The Access Server does not check for duplicate information.) Select this option if the application expects to receive information for all instances of the action. In this case, the application must process the values of all duplicate actions returned to it. Use of this option may incur performance issues.
- **ACTION_IGNORE—**If you chose this option, the Access Server removes all duplicate actions, and only the first instance of the action is returned to the application requesting authorization for the user. Each time an action value is added, the Access Server checks existing values to determine if the new action duplicates an existing one. If the Access Server finds one, it does not add the new value to those it returns to the application. In this case, any information inherent to the value of the repeated action is lost. Because the Access Server

must check for duplicate actions, use of this option may incur performance costs

- **ACTION_OVERWRITE—**If you choose this option, only the value of the last instance of the action is returned. Each new value overwrites the previous one, and previous values are lost. Do not select this option if the application requesting the authorization expects the results of all duplicate actions. This option is the most efficient one.

Duplicate Actions and WebGate Restrictions—The ability to process duplicate actions applies to AccessGates only. The Access Server sends to the WebGate the actions as specified by the duplicate actions policy—whether Duplicate, Ignore Duplicate, or Overwrite. However, the WebGate supports only a single value per header variable. Although it receives the duplicate actions, the WebGate overrides duplicates such that the last value set for the header variable is used. Values set for the same header variable by previous actions are lost.

Label	Type	Range	Description
Authorization Expression	A string Required	Any	An expression containing one or more authorization rules, specified by name, and the symbols used to combine them.
Name (key)	A string Required	Any	A unique name.
Description	A string Optional	Any	A description.
Enabled	A Boolean flag Required	0, meaning no, or other, meaning yes.	Answers the question: Is the rule enabled?
Duplicate Actions Policy	A string Optional	Any one of the following constants: ACTION_DUPICATE = 0 (Duplicate) ACTION_IGNORE = 1 (Ignore) ACTION_OVERWRITE = 2 (Overwrite) UNDEFINED = 3	The policy for the Access Server to follow if it encounters duplicate actions as a result of evaluation of this expression.

Label	Type	Range	Description
Success Actions	An array of Action objects Optional	See the class definition on page 488.	Actions to be performed by the client if authorization succeeds as a result of evaluation of the expression. The ObAmObjectWithActions class is used to manage success actions.
Failure Actions	An array of Action objects Optional	See the class definition on page 488.	Actions to be performed by the client if the authorization fails and the user is denied access as a result of evaluation of the expression. The ObAMObjectWithActions class is used to manage failure actions.
Inconclusive Actions	An array of Action objects Optional	See the class definition on page 488.	Actions to be performed by the client if authorization cannot be conclusively determined as a result of evaluation of the expression. In this case, the user is denied access to the resource. The ObAmObjectWithActions class is used to manage success actions.

Java

```
public ObAMAuthorizationExpr();
public String getExpression();
public void setExpression(String value)
        throws ObAMException;
public int getDuplicateActionsPolicy();
public void setDuplicateActionsPolicy(int value)
        throws ObAMException;
public int getNumberOfActions(int actionType)
        throws ObAMException;
public ObAMAction getActionOfType(int actionType,
        int index) throws ObAMException;
public void addActionOfType(int actionType,
        ObAMAction value) throws ObAMException;
public void removeActionOfType(int actionType,
        ObAMAction value) throws ObAMException;
public String getName();
public void setName(String value);
public void setIDFrom(ObAMObjectWithActions other);
```

C

```
typedef const void * ObAMAuthorizationExpr_t;
ObAMAuthorizationExpr_t ObAMAuthorizationExpr_new();
ObAMAuthorizationExpr_t ObAMAuthorizationExpr_copy(
        ObAMAuthorizationExpr_t authz);
void ObAMAuthorizationExpr_delete(
        ObAMAuthorizationExpr_t *pAuthz);
const char *ObAMAuthorizationExpr_getExpr(
        ObAMAuthorizationExpr_t authz);
int ObAMAuthorizationExpr_getDuplicateActionsPolicy(
        ObAMAuthorizationExpr_t authz);
int ObAMAuthorizationExpr_getNumberOfActions(
        ObAMObjectWithActions_ActionType type,
        ObAMAuthorizationExpr_t authz);
ObAMAction_tObAMAuthorizationExpr_getActionOfType(
        ObAMObjectWithActions_ActionType type,
        ObAMAuthorizationExpr_t authz, int index);
void ObAMAuthorizationExpr_setIDFrom(
        ObAMAuthorizationExpr_t authz,
        ObAMAuthorizationExpr_t other);
void ObAMAuthorizationExpr_setExpr(
        ObAMAuthorizationExpr_t authz, const char *value);
voidObAMAuthorizationExpr_setDuplicateActionsPolicy(
        ObAMAuthorizationExpr_t authz,
        ObAMAuthorizationExpr_DuplicateActionsPolicy value);
```

Creating an Authorization Expression

Listing 40 shows a portion of an example program. This code creates a default authorization expression for a policy domain. It performs the following functions:

- It creates an ObAMAuthorizationExpr object called pd_expr and specifies the content of the expression—that is, the authorization rules of the expression and how they are combined in the expression.

```
ObAMAuthorizationExpr pd_expr =
                        new ObAMAuthorizationExpr();
pd_expr.setExpression ("Authz Rule 1 & Authz Rule 2");
```

- It sets the duplicate actions policy for the expression to OVERWRITE.

```
pd_expr.setDuplicateActionsPolicy(
              ObAMAuthorizationExpr.ACTION_OVERWRITE);
```

- It creates an ObAMAction object to be used for an action. Using the object, it defines an action to be taken if the result of evaluation of the expression is Inconclusive.

```
ObAMAction action3 = new ObAMAction();
action3.setType("otherType");
action3.setName("authzAction");
action3.setValue("a");
action3.setValueType(ObAMAction.FIXEDVALUE);
```

- It adds the Inconclusive action to the authorization expression

```
pd_expr.addActionOfType(
                ObAMObjectWithActions.INCONCLUSIVE,
                action3);
```

- It adds the authorization expression to the policy domain.

```
domain.setDefaultAuthorizationExpr(pd_expr);
```

Listing 40 Creating an Authorization Expression

```
                        .
                        .
                        .
ObAMAuthorizationExpr pd_expr = new ObAMAuthorizationExpr();
pd_expr.setExpression("Authz Rule 1 & Authz Rule 2");
pd_expr.setDuplicateActionsPolicy(
                        ObAMAuthorizationExpr.ACTION_OVERWRITE);
ObAMAction action3 = new ObAMAction();
action3.setType("otherType");
action3.setName("authzAction");
action3.setValue("a");
action3.setValueType(ObAMAction.FIXEDVALUE);
pd_expr.addActionOfType(
                        ObAMObjectWithActions.INCONCLUSIVE, action3);
domain.setDefaultAuthorizationExpr(pd_expr);
                        .
                        .
                        .
```

Class ObAMAuthorizationExprMgd

(Managed Code). This class defines a managed authorization expression. It specifies the conditions for allowing or denying access to resources covered by a policy or policy domain.

Managed Code Form

```
     __property System::String *get_Expr();
__property int get_DuplicateActionsPolicy();
int getNumberOfActions(ObAMActionTypeMgd *type);
ObAMActionMgd *getActionOfType(ObAMActionTypeMgd *type,int index);
__property void set_Expr(System::String *value);
void setDuplicateActionsPolicy(ObDuplicationActionPolicyMgd *value);
void addActionOfType(ObAMActionTypeMgd *type, ObAMActionMgd *value);
void modifyActionOfType(ObAMActionTypeMgd *type,ObAMActionMgd *value);
void removeActionOfType(ObAMActionTypeMgd *type,ObAMActionMgd *value);
```

Class ObAMDuplicateActionPolicyMgd

(Managed Code). Class used to describe the policy for dealing with duplicate actions. The choices are to duplicate, ignore, overwrite, or undefined.

Managed Code Form

```
void setDuplicate();
void setIgnore();
void setOverWrite();
void setUndefined();
```

Class ObAMAccessConditions

An object of the ObAMAccessConditions class specifies for an authorization rule the conditions under which access is allowed or denied to the protected resource.

Label	Type	Range	Description
Role (no key)	An array of strings At least one required	One of: none anyone noone	A set of roles
Person	An array of Identity objects Optional	See the class definition on page 494.	A set of people against which the user requesting access is compared
Group	An array of Identity objects Optional	See the class definition on page 494.	A set of groups against which the user is compared
Rule	An array of strings At least one required	Any	A set of LDAP rules that select user profiles
IPAddress	An array of strings At least one required	Any	A set of IP addresses against which the user's IP address is compared

Java

```
public int getNumberOfPersons();
public int getNumberOfGroups();
public int getNumberOfRoles();
public int getNumberOfRules();
public int getNumberOfIPaddresses();
public ObAMIdentity getPerson(int index);
public ObAMIdentity getGroup(int index);
public String getRole(int index);
```

```
   public String getRule(int index);
   public String getIPaddress(int index);
   public void addRole(String value);
   public void addPerson(ObPerson value);
   public void addGroup(ObGroup value);
   public void addRule(String value);
   public void addIPAddress(String value);
   public void removeRole(String value);
   public void removePerson(ObPerson value);
   public void removeGroup(ObGroup value);
   public void removeRule(String value);
   public void removeIPAddress(String value);
   public void setIDFrom(ObAMAccessConditions other);
```

Class ObAMAccessConditionsMgd

(Managed Code). An ObAMAccessConditionsMgd object specifies the conditions under which access is allowed or denied in an authorization rule. For details on object components, see “Class ObAMAccessConditions” on page 478.

Managed Code

```
   // Getters and setters
   _property int get_NumberOfPersons();
_property int get_NumberOfGroups();
_property int get_NumberOfRoles();
_property int get_NumberOfRules();
_property int get_NumberOfIPaddresses();
ObAMIdentityMgd *getPerson (int index);
ObAMIdentityMgd *getGroup (int index);
System::String *getRole(int index);
System::String *getRule(int index);
System::String *getIPaddress(int index);
_property void set_IDFrom(ObAMAccessConditionsMgd *access);
_property void set_AddRole(System::String *value);
_property void set_AddPerson(ObAMIdentityMgd *value);
_property void set_AddGroup(ObAMIdentityMgd *value);
_property void set_AddRule(System::String *value);
_property void set_AddIPaddress(System::String *value);
_property void set_RemoveRole(System::String *value);
_property void set_RemovePerson(ObAMIdentityMgd *value);
_property void set_RemoveGroup(ObAMIdentityMgd *value);
_property void set_RemoveRule(System::String *value);
_property void set_RemoveIPaddress(System::String *value);
```

Class ObAMActionTypeMgd

(Managed Code). Class used to describe the type of action being requested. The set methods are used to define the type of action requested.

Managed Code

```
void setSuccess();
void setFailure();
void setInconclusive();
```

Class ObAMObjectWithActions

The ObAMObjectWithActions class is an abstract class that is inherited by the ObAMAuthenticationRule class, the ObAMAuthorizationRule class, and the ObAMAuthorizationExpr class when they are instantiated. Do not instantiate this class directly.

You use the methods of this class to get and set information about actions for objects of any of these other classes. The ObAMObjectWithActions class methods allow you to specify the kind of action you are interested in. The class defines an enumeration to specify the three types of actions: Success, Failure, and Inconclusive.

Label	Type	Range	Description
Name	String, Optional	any	The name of the authorization expression object

Label	Type	Range	Description
Action Type	An enumerated value, Integer	SUCCESS = 0 FAILURE = 1 INCONCLUSIVE = 2 See the class definition on page 488.	A value that identifies the type of action that the method applies to. The types apply in the following way: • For ObAMAuthenticationRule An authentication rule can have associated with it SUCCESS actions and FAILURE actions. • For ObAMAuthorizationRule An authorization rule can have associated with it SUCCESS actions and FAILURE actions. • For ObAMAuthorizationExpr An authorization expression can have associated with it SUCCESS, FAILURE, and INCONCLUSIVE actions.

Java

```
public int getNumberOfActions(int actionType)
        throws ObAMException;
public ObAMAction getActionOfType(int actionType, int index)
        throws ObAMException;
public void addActionOfType (int actionType,
        ObAMAction value) throws ObAMException;
public void removeActionOfType(int actionType,
        ObAMAction value) throws ObAMException;
public String getName();
public void setName(String value);
public void setIDFrom(ObAMObjectWithActions other);
```

Obsoleted Methods

The ObAMObjectWithActions class is a new class provided with NetPoint 6.5. Its methods replace the following ones used in previous releases.

Do not use following methods. They have been obsoleted as of NetPoint Access Manager 6.5:

```
public ObAMAction getSuccessAction (int index);
public ObAMAction getFailureAction (int index);
public int getNumberOfSuccessActions();
public int getNumberOfFailureActions();
public void addSuccessActions(ObAMAction value);
```

```
public void addFailureActions(ObAMAction value);
public void removeSuccessActions(ObAMAction value);
public void removeFailureActions(ObAMAction value);
```

Class ObAMTimingConditions

An authorization rule can include timing conditions that set the period of time when the rule is in effect. If you do not set a timing condition, by default the authorization rule is always in effect. Timing conditions affect both the Allow Access and the Deny Access conditions of the rule.

An object of the ObAMTimingConditions class contains timing conditions specifying when the rule is in effect.

Label	Type	Range	Description

RelativeTo
 	
(no key)
	Integer
	Optional
		`UNDEFINED=0`
		`GMT=1`
		`LOCAL_TIME=2`
			A flag indicating whether the timing conditions are relative to GMT or local time

StartDate	A single Date object Optional	See the class definition on page 493.	A start date for the period during which the Timing Condition applies
StartTime	A single Time object Optional	See the class definition on page 499.	A start timefor the period during which the Timing Condition applies
EndDate	A single Date object Optional	See the class definition on page 493.	An end date for the period during which the Timing Condition applies
EndTime	A single Tme object Optional	See the class definition on page 499.	An end time for the period during which the Timing Condition applies
Months	An array of strings Optional	See the list of valid month values under Date objects on page 493.	Sets of months
DayofMonth	An array of strings Optional	See the list of valid day values under Date objects on page 493.	Sets of days of the month
DayofWeek	An array of strings Optional	See the discussion for this value under Date objects on page 493.	Sets of days of the week.

Java

```
public int getRelativeTo();
```

```
public ObAMDate getStartDate();
public ObAMTime getStartTime();
public ObAMDate getEndDate();
public ObAMTime getEndTime();
public int getNumberOfMonths();
public int getNumberOfDaysOfMonth();
public int getNumberOfDaysOfWeek();
public int getMonth(int index);
public int getDayOfMonth(int index);
public int getDayOfWeek(int index);
public void setRelativeTo(int value)
        throws ObAMException;
public void setStartDate(ObAMDate value);
public void setStartTime(ObAMTime value);
public void setEndDate(ObAMDate value);
public void setEndTime(ObAMTime value)
public void addMonth(int value)
        throws ObAMException;
public void addDayOfMonth(int value)
        throws ObAMException;
public void addDayOfWeek(int value)
        throws ObAMException;
public void removeMonth(int value)
        throws ObAMException;
public void removeDayOfMonth(int value)
        throws ObAMException;
public void removeDayOfWeek(int value)
        throws ObAMException;
public void setIDFrom(ObAMTimingConditions other);
```

The code in Listing 41 creates an authorization rule and sets the timing conditions for it. The code performs the following functions:

- It creates an ObAMAuthorizationRule object called authzRule2, sets the name member of the object to Authz Rule 2, and it enables the rule.

```
ObAMAuthorizationRule authzRule2 = new
                        ObAMAuthorizationRule();
authzRule2.setName("Authz Rule 2");
authzRule2.setEnabled(true);
```

- It creates an ObAMTimingConditions object called timing2 to be used for the rule's timing conditions.

```
ObAMTimingConditions timing2 = new
                        ObAMTimingConditions();
```

- It creates an ObAMDate object called startDate to specify the date beginning from which the rule is applicable. It creates an ObAMTime object to specify the time beginning from which the rule applies. It sets the starting date and it sets the starting time in these objects.

```
ObAMDate startDate = new ObAMDate();
ObAMTime startTime = new ObAMTime();
```

```
startDate.set(2001, ObAMDate.OCTOBER, 31);
startTime.set(12, 0, 0);
```

- It specifies that the time is relative to the local time on the Web server.

```
timing2.setRelativeTo(ObAMTimingConditions.LOCAL_TIME);
```

The remainder of the code sets the timing conditions to allow anyone access after 12:00:00 of October 15, 2001 on the 1st, 2nd, and 30th of January and November if the day is either a Monday or a Tuesday.

Listing 41 Creating an Authorization Rule with Timing Conditions

```
                    .
                    .
                    .
ObAMAuthorizationRule authzRule2 = new ObAMAuthorizationRule();
authzRule2.setName("Authz Rule 2") ;
authzRule2.setEnabled(true);
ObAMTimingConditions timing2 = new ObAMTimingConditions();
ObAMDate startDate = new ObAMDate();
ObAMTime startTime = new ObAMTime();
startDate.set(2001, ObAMDate.OCTOBER, 31);
startTime.set(12, 0, 0);
timing2.setRelativeTo(ObAMTimingConditions.LOCAL_TIME);
timing2.setStartDate(startDate);
timing2.setStartTime(startTime);
timing2.addMonth(ObAMDate.JANUARY);
timing2.addMonth(ObAMDate.NOVEMBER);
timing2.addDayOfMonth(1);
timing2.addDayOfMonth(2);
timing2.addDayOfMonth(30);
timing2.addDayOfWeek(ObAMDate.MONDAY);
timing2.addDayOfWeek(ObAMDate.TUESDAY);
authzRule2.setTimingConditions(timing2);
                .
                .
                .
```

Class ObAMTimingConditionsMgd

An object of the ObAMTimingConditions class contains timing conditions to be set for an authorization rule to specify when that rule is in effect.

Label	Type	Range	Description
StartDate	A single Date object Optional	See the class definition on page 493.	A start date for the period during which the Timing Condition applies
StartTime	A single Tme object Optional	See the class definition on page 499.	A start timefor the period during which the Timing Condition applies
EndDate	A single Date object Optional	See the class definition on page 493.	An end date for the period during which the Timing Condition applies
EndTime	A single Tme object Optional	See the class definition on page 499.	An end time for the period during which the Timing Condition applies
Months	An array of strings Optional	See the list of valid month values under Date objects on page 493.	Sets of months
DayofMonth	An array of strings Optional	See the list of valid day values under Date objects on page 493.	Sets of days of the month
DayofWeek	An array of strings Optional	See the discussion for this value under Date objects on page 493.	Sets of days of the week.

In addition to the other settings for timing conditions, the class ObAMTimingConditionsMgd indicates whether the timing conditions are relative to GMT or local time. See “Class ObAMTimingConditions_RelativeToMgd” on page 487 for details.

Managed Code Form

```
// Getters and setters
_property ObAMDateMgd *get_StartDate();
_property ObAMTimingConditions_RelativeToMgd *get_RelativeTo();
_property ObAMTimeMgd *get_StartTime();
_property ObAMTimeMgd *get_EndDate();
```

```
_property ObAMTimeMgd *get_EndTime();
_property int get_NumberOfMonths();
_property int get_NumberOfDaysOfMonth();
_property get_NumberofDaysOfWeek();
int getMonth(int index);
int getDayOfMonth(int index);
int getDayOfWeek(int index);
_property void set_IDFrom(ObAMTimingConditionsMgd *other);
_property void set_RelativeTo(
                        ObAMTimingConditions_RelativeToMgd*value);
_property void set_StartDate(ObAMDateMgd *date);
_property void set_EndTime(ObAMTimeMgd *time);
_property void set_EndDate(ObAMDateMgd *date);
_property void set_StartTime(ObAMDateMgd *time);
_property void set_AddMonth(ObAMDate_MonthsMgd *value);
_property void set_AddDayOfMonth(int value);
_property void set_AddDayOfWeek(ObAMDate_DaysOfWeekMgd *value);
_property void set_RemoveMonth(ObAMDate_MonthsMgd *value);
_property void set_RemoveDayOfMonth(int value);
_property void set_RemoveDayOfWeek(ObAMDate_DaysOfWeekMgd *value);
```

Class ObAMTimingConditions_RelativeToMgd

(Managed Code). This class defines a managed value type for defining the various timing conditions used by the ObAMTimingConditionsMgd class. This class provides a wrapper around the enum ObAMTimingConditions_RelativeTo. You can either set the value using the setter methods, or you can use an ObAMTimingConditions_RelativeTo value.

The methods for this class are as follows:

```
setUndefined() = UNDEFINED
setGMT() = GMT
setLocalTime() = LOCAL_TIME
```

Managed Code Form

```
// Getters and setters
_property bool get_isUndefined();
_property bool get_isGMT();
_property bool get_isLocalTime();
_property ObAMTimingConditions_RelativeTo get_Value();
void setUndefined();
void setAsGMT();
void setAsLocalTime();
```

Class ObAMDate_DaysOfWeekMgd

(Managed Code). This is a wrapper around unmanaged enum of days of the week. Use this class to define a day of the week when setting timing conditions.

Managed Code Form

```
ObAMDate_DaysOfWeekMgd();
   void setSunday();
   void setMonday();
   void setTuesday();
   void setWednesday();
   void setThursday();
   void setFriday();
   void setSaturday();
```

Class ObAMAction

An object of the ObAMAction class represents an action to be returned on success or failure of an authentication or authorization process. The action is interpreted by a WebGate or other Access Client.

Label	Type	Range	Description

Type
 	
(key1)
	A string
	Required
		Any of the values:
		
		`redirectURL`
		`headerVars`
		`custom`
			A type indicating what action should be taken

Name (key2)	A string Required	Any	The action name
Value	A string Required	Any	The action value
ValueType	A string Required	Any	The type of the value, which can be a fixed value or a user attribute

Java

```
public String getType();
public String getName();
public String getValue();
public int getValueType();
public void setType(Stromg value);
public void setName(String value);
public void setValue(String value);
```

```
public void setValueType(int value)
        throws ObAMException;
public void setIDFrom(ObAMAction other);
```

Class ObAMActionMgd

(Managed Code). An ObAMActionMgd class represents an action to be performed on a successful or failed authentication or authorization. The action is interpreted by a WebGate or other Access Client.

Label	Type	Range	Description

Type
 	
(key1)
	A string
	Required
		Any of the values:
		
		`redirectURL`
		`headerVars`
		`custom`
			A type indicating what action should be taken

Name (key2)	A string Required	Any	The action name
Value	A string Required	Any	The action value
ValueType	A string Required	Any	The type of the value, which can be a fixed value or a user attribute

Managed CodeForm

```
// Getters and Setters
     _property System::String *get_Type();
     _property System::String *get_Name();
     _property System::String *get_value();
     _property ObAMAction_ValueTypeMgd *get_ValueType();
     _property void set_IDFrom(ObAMActionMgd *other);
     _property void set_Type(System::String *value);
     _property void set_Name(System::String *value);
     _property void set_Value(System::String *value);
     _property void set_ValueType(ObAMAction_ValueTypeMgd *value);
```

Class ObAMAction_ValueTypeMgd

(Managed Code). An ObAMAction_ValueTypeMgd object represents the value types used by the ObAMActionMgd class. This class provides a wrapper around the enum ObAMAction_ValueType.

Managed CodeForm

```
// Getters and Setters
      _property bool get_isUndefined();
      _property bool get_isFixedValue();
      _property bool get_isAttribute();
      _property obAMAction_ValueType get_Value();
      _property void set_Value(ObAMAction_ValueType value);
      void setUndefined();
      void setFixedValue();
      void setAttribute();
```

Class ObAMAuditRule

An object of the ObAMAuditRule class defines the kind of auditing to be done for a policy or policy domain.

Label	Type	Range	Description
Events (no key)	An array of strings At least one required	See the list provided for Master Audit Rule on page 443.	A set of events for which audit records will be generated
Attributes	An array of strings At least one required	Any	A set of user profile attributes to be included in the audit records

Java

```
public int getNumberOfEvents();
public int getNumberOfAttributes();
public int getEvent(int index);
public String getAttribute(int index);
public void addEvent(int value);
public void addAttribute(String value);
public void removeEvent(int value);
public void removeAttribute(String value);
public void setIDFrom(ObAMAuditRule other);
```

Class ObAMAuditRuleMgd

(Managed Code). An ObAMAuditRuleMgd object specifies how auditing is to be done for a policy or policy domain.

Label	Type	Range	Description
Events (no key)	An array of strings At least one required	See the list provided for Master Audit Rule on page 443.	A set of events for which audit records will be generated
Attributes	An array of strings At least one required	Any	A set of user profile attributes to be included in the audit records

Managed Code Form

```
// Getters and setters
__property int get_NumberOfEvents();
__property int get_NumberOfAttributes();
ObAMAuditRule_EventTypeMgd *getEvent(int index);
System::String *getAttribute(int index);
__property void set_IDFrom(ObAMAuditRuleMgd *other);
__property void set_AddEvent(ObAMAuditRule_EventTypeMgd *value);
__property void set_AddAttribute(System::String *value);
__property void set_RemoveEvent(ObAMAuditRule_EventTypeMgd *value);
__property void set_RemoveAttribute(System::String *value);
```

Class ObAMAuditRule_EventTypeMgd

(Managed Code). This is a Managed Value Type class that defines the event types used by the ObAuditRuleMgd class. This class provides a wrapper around the enum ObAMAuditRule_eventType. You may either set the value using the setter methods, or use an ObAMAuditRule_EventType value.

Managed Code Form

```
// Get and set values
__property bool get_isUndefined();
__property bool get_isAuthenticationSuccess();
__property bool get_isAuthenticationFailure();
__property bool get_isAuthorizationSuccess();
__property bool get_isAuthorizationFailure();
__property ObAMAuditRule_EventType get_Value();
__property void set_Value(ObAMAuditRule_EventType value);
void setUndefined();
void setAuthenticationSuccess();
void setAuthenticationFailure();
void setAuthorizationSuccess();
```

```
void setAuthorizationFailure();
```

Class ObAMDate

An object of the ObAMDate class represents a date. The ObAMDate class includes one set method to set the year, month, and day. If the date or time is invalid, the method throws an ObAMException with the BAD_OBJECT code.

Label	Type	Range	Description
Year (no key)	Integer Required	The full year	An integer representing the year value in the object
Month	Integer Required	One of an enumerated list, as follows: JANUARY=1 FEBRUARY=2 MARCH=3 APRIL=4 MAY=5 JUNE=6 JULY=7 AUGUST=8 SEPTEMBER=9 OCTOBER=10 NOVEMBER=11 DECEMBER=12	An integer representing the month value in the object
Day	Integer Required	One of an enumerated list, as follows: SUNDAY=1 MONDAY=2 TUESDAY=3 WEDNESDAY=4 THURSDAY=5 FRIDAY=6 SATURDAY=7	An integer representing the day value in the object

Java

```
public int getYear();
public int getMonth();
public int getDay();
```

Class ObAMDateMgd

(Managed Code) An ObAMDateMgd object represents a date. The ObAMDateMgd class has one set method to set the year, month, and day. If the date or time is invalid, set will throw an ObAMException with the BAD_OBJECT code. For details on the object components, see “Class ObAMDate” on page 492

Managed Code Form

```
// Getters and setters
_property int get_Year();
_property int get_Month();
_property int get_Day();
void set(int year, int month, int day_);
```

Class ObAMDate_MonthsMgd

Use this class to define months of the year when setting timing conditions.

Managed Code Form

```
ObAMDate_MonthsMgd();
void setJanuary();
void setFebruary();
void setMarch();
void setApril();
void setMay();
void setJune();
void setJuly();
void setAugust();
void setSeptember();
void setOctober();
void setNovember();
void setDecember();
```

Class ObAMDate_DaysOfWeekMgd

Use this class to define days of the week when setting the timing conditions (using ObAMATimingConditionsMgd class) or setting the date (using the ObAMADateMgd class).

Managed Code Form

```
ObAMDate_DaysOfWeekMgd();
void setSunday();
void setMonday();
void setTuesday();
void setWednesday();
void setThursday();
void setFriday();
void setSaturday();
```

Class ObAMIdentity

An object of the ObAMIdentity class identifies a user or group profile in the user directory used in an access condition or an admin rule.

Label	Type	Range	Description
UID (no key)	A string Required	Any	A unique ID for the identity
Name	A string Required	Any	A name
LoginID	A string Required	Any	A login ID, as defined by the person object class in COREid. Empty for group identities.

Java

```
public String getUID();
public String getName();
public String getLoginID();
public void setUID(String value);
public void setName(String value);
public void setLoginID(String value);
public void serialize(String value);
```

Note: For an Identity object, the Name member is strictly a label and does not correspond to a directory entry. For that reason, an `ObAMIdentityObject` for which only the `Name` has been set cannot be added to any other object.

Class ObAMIdentityMgd

(Managed Code). An ObAMIdentityMgd object identifies a user or group profile in the user directory used in an access condition or an admin rule.

Label	Type	Range	Description
UID (no key)	A string Required	Any	A unique ID for the identity
Name	A string Required	Any	A name
LoginID	A string Required	Any	A login ID, as defined by the person object class in COREid. Empty for group identities.

Managed Code Form

```
// Getters and setters
   _property System::String *get_UID();
   _property System::String *get_Name();
   _property System::String *get_LoginID();
   _property ObAMIdentity *get_UnmanageIdentity();
   _property void set_UID(System::String *value);
   _property void set_Name(System::String *value);
   _property void set_LoginID(System::String *value);
```

Note: Unique to the Identity object, the Name member is strictly a label and does not correspond to a directory entry. For that reason, an `ObAMIdentityObjectMgd` for which only the `Name` has been set cannot be added to any other object.

Class ObAMParameter

An object of the ObAMParameter class supplies a name-value pair to be matched against request data (for example, an HTTP query string or POST data) when determining if a policy applies to an access request. ObAMParameter objects are also used in authorization schemes.

Label	Type	Range	Description
Name (key)	A string Required	Any	A name
Value	A string Required	Any	A value for the name

Java

```
public String getName();
public String getValue();
public void setName(String value);
public void setValue(String value);
```

Class ObAMParameterMgd

(Managed Code) An ObAMParameterMgd object supplies a name-value pair to be matched against request data (for example, an HTTP query string or POST data) when determining if a policy applies to an access request. ObAMParameterMgd objects are also used in authorization schemes. For details on the object components, see “Class ObAMParameter” on page 496.

Managed Code

```
// Getters and setters
_property System::String *get_Value();
_property System::String *get_Name();
_property void set_Name(System::String *value);
_property void set_Value(System::String *value);
```

Class ObAMResource

An object of the ObAMResource class represents a set of resources to which policy domains or policies apply. The resource set is selected by matching components of the Resource URL to ObAMResource members.

Label	Type	Range	Description
ResourceType (key1)	A string Required	Any	A built-in or custom resource type
HostID (key2)	A string Required	Any	A host ID that specifies a set of hostnames, IP address, and ports
URLPrefix (key3)	A string Required	Any	A URL prefix that matches the initial local part of the URL
Description	A string Optional	Any	An optional description

Java

```
public String getResourceType();
public String getHostID();
public String getURLPrefix();
public String getDescription();
public void setResourceType(String value);
public void setHostID(String value);
public void setURLPrefix(String value);
public void setDescription(String value);
public void setIDFrom(ObAMResource other);
```

Class ObAMResourceMgd

(Managed Code). An ObAMResourceMgd object represents a set of resources to which policy domains or policies apply. The resource set is selected by matching components of the Resource URL to ObAMResourceMgd members. For details on the object components, see “Class ObAMResourceMgd” on page 497.

Managed Code

```
// Getters and setters
_property System::String *get_ResourceType();
_property System::String *get_HostID();
_property System::String *get_URLPrefix();
_property System::String *get_Description();
_property void set_IDFrom(ObAMResourceMgd *other);
```

```
_property void set_ResourceType(System::String *value);
_property void set_HostID(System::String *value);
_property void set_URLPrefix(System::String *value);
_property void set_Description(System::String *value);
```

Class ObAMTime

An object of the ObAMTime class represents a specific time. The ObAMTime class has one set method to set the hour, minutes and seconds. If the values provided are invalid, set will throw an ObAMException with the BAD_OBJECT code.

Label	Type	Range	Description
Hours (no key)	Integer Required	0 to 23	An integer representing the hour value in the object
Minutes	Integer Required	0 to 59	An integer representing the minutes value in the object
Seconds	Integer Required	0 to 59	An integer representing the seconds value in the object

Java

```
public int getHours();
public int getMinutes();
public int getSeconds();
public void set(int hours, int minutes, int seconds)
                        throws ObAMException;
```

Class ObAMTimeMgd

(Managed Code). An ObAMTimeMgd object represents a time. The ObAMTimeMgd class has one set method to set the hour, minutes and seconds. If the values provided are invalid, set will throw an ObAMException with the BAD_OBJECT code. For details on the object components, see “Class ObAMTime” on page 498.

Label	Type	Range	Description
Hours (no key)	Integer Required	0 to 23	An integer representing the hour value in the object
Minutes	Integer Required	0 to 59	An integer representing the minutes value in the object
Seconds	Integer Required	0 to 59	An integer representing the seconds value in the object

Managed Code

```
// Getters and setters
_property int get_Hours();
_property int getMinutes();
_property int get_Seconds();
void set(int hours, int minutes, int seconds);
```

Test Objects

Test objects are objects that are used to provide input to and capture output from the ObAccessManager testAccess method. Because none of these objects is intended to be stored, they contain no key fields. Here are the three test objects:

- “Class ObAMAccessTest” on page 499
- “Class ObAMAccessTestResults” on page 505
- “Class ObAMAccessTestResult” on page 506

Class ObAMAccessTest

An object of the ObAMAccessTest class represents a test of NetPoint access policies. It specifies a resource and one or more users for whom access to the resource is to be tested.

The ObAMAccessTest object is the input argument to the ObAccessManager testAccess method, which returns an ObAMAccessTestResults object with the results of the access test.

Label	Type	Range	Description
URL (no key)	A string Required	Any	The URL (minus the resource type)
Resource	A string Required	Any	The resource type
Operation	An array At least one required	Any	The list of operations (defined by the resource type)
IPAddress	A string Optional	Any	An IP address
Date	One Date object Optional	See the class definition on page 492.	A date
Time	One Time object Optional	See the class definition on page 498.	A time
User	An array of Identity objects Optional	See the class definition on page 494.	An array of users. If omitted, access is tested for all users in the user directory
ShowAllowed	A Boolean flag, Required	0, meaning no, or other, meaning yes.	Specifies if the test should show users that are allowed accesses.
ShowDenied	A Boolean flag, Required	0, meaning no, or other, meaning yes.	Specifies if the test should show users that are denied accesses
ShowMatchingPolicy	A Boolean flag, Required	0, meaning no, or other, meaning yes.	Specifies if the test should show the policies that apply for each user
ShowMatchingExpr	A Boolean flag, Required	0, meaning no, or other, meaning yes.	Specifies if the test should show the authorization expression that applies for each user

Label	Type	Range	Description
ShowDeterminingRules	A Boolean flag, Required.	0, meaning no, or other, meaning yes.	Specifies if the test should show the authorization rules that were the determining rules of the authorization expression. For a description of determining rules, see “Class ObAMAuthorizationExpr” on page 471.

Java

```
public String getURL();
public String getResourceType();
public String getIPaddress();
public ObAMDate getDate();
public ObAMTime getTime();
int getNumberOfOperations();
int getNumberOfUsers();
String getOperation(int index);
public boolean getShowMatchingPolicy();
public boolean getShowMatchingExpr();
public boolean getShowDeterminingRules();
ObAMIdentity getUser(int index);
public boolean getShowAllowed();
public boolean getShowDenied();
public void setURL(String value);
public void setResourceType(String value);
public void addOperation(String value);
public void setIPaddress(String value);
public void setDate(ObAMDate value);
public void setTime(ObAMTime value);
public void addUser(ObAMIdentity value);
public void setShowAllowed(boolean value);
public void setShowDenied(boolean value);
public void setShowMatchingPolicy(boolean value);
public void setShowMatchingExpr(boolean value);
public void setShowDeterminingRules(boolean value);
```

Obsoleted Methods

Do not use following methods. They have been obsoleted as of NetPoint Access Manager 6.5:

```
getShowMatchingRule()
setShowMatchingRule();
```

Setting Up a Test Using Two Users

Listing 42 shows an excerpt from a sample program which is delivered with NetPoint 6.5 and later. The code sets up a test to check authorization for two users, and it displays the results. The code uses an object of the ObAMAccessTest class to set up the test and an object of the ObAMAccessTestResults class for the results. See “Class ObAMAccessTestResults” on page 505 for details. The code creates two ObAMIdentity objects, one for each user for whom authorization will be checked. It defines the policy to be used for the test, and it specifies the information to be returned for each user, including the result of evaluation of the authorization expression—whether authorization succeeded, failed, or whether a processing error occurred. For each user, it prints out the test information, including the determining rules that led up to the outcome of the expression result. Listing 42 shows the code segment.

Listing 42 Testing the System for Authorization Using Two Users

```
static void example5(ObAccessManager am) throws ObAMException
{                       .
                        .


        ObAMAccessTest test = new ObAMAccessTest();
        ObAMIdentity person1 = new ObAMIdentity();
        ObAMIdentity person2 = new ObAMIdentity();

        test.setResourceType("http");
        test.setURL("host1/myresources/doc1.html")
        test.addOperation("GET");
        test.addOperation("POST");
        test.setIPaddress("192.168.1.14");
        ObAMDate date = new ObAMDate();
        ObAMTime time = new ObAMTime();
        date.set(2001, ObAMDate.NOVEMBER, 15);
        time.set(12, 0, 0);
        test.setDate(date);
        test.setTime(time);

        person1.setLoginID("A.Loomis");
        person2.setLoginID("J.Himes");
        person1.setLoginID("admin")
        test.addUser(person1);
        test.addUser(person2);

        test.setShowAllowed(true);
        test.setShowDenied(true);
        test.setShowMatchingPolicy(true);
        test.setShowMatchingExpr(true);
        test.setShowDeterminingRules(true);
```

```
    ObAMAccessTestResults results = am.testAccess(test);
    System.out.println("Policy Domain : " +
            results.getPolicyDomain());
    for (int i = 0; i < results.getNumberOfResults(); i++) {
            ObAMAccessTestResult result = results.getResult(i);
            System.out.println("Result:");
            System.out.println("User : " + result.getUser().getUID());
    if (result.getAuthorized() == true)
    {
         System.out.println("Authorized : ALLOWED");
    }
    else
    {
    if (result.getAuthorizationStatus() ==
                            ObAMAccessTestResult.DENIED)
            {
                 System.out.println("Authorized : DENIED");
            }
    else if (result.getAuthorizationStatus() ==
                        ObAMAccessTestResult.INCONCLUSIVE)
            {
                 System.out.println
                              ("Authorized : INCONCLUSIVE");
            }
            else
            {
                 System.out.println("Authorized : ERROR");
    }
}
                 System.out.println("Policy : " + result.getPolicy());
                 System.out.println(" Expr       : " + result.getExpr());
                 System.out.println("Determining Rules: ");
            for (int j=0; j<result.getNumberOfDeterminingRules(); j++)
       {
            System.out.println(" " +result.getDeterminingRule(j)) ;
       }
            }
    }
```

Class ObAMAccessTestMgd

(Managed Code). An object of the ObAMAccessTestMgd class represents a test of NetPoint access policies. It specifies a resource request and one or more users for whom access to the resource is to be tested. The ObAMAccessTestMgd object is the argument to the ObAccessManager testAccess method, which returns an ObAMAccessTestResultsMgd object with the results of the access test for each user. For details on the object components, see “Class ObAMAccessTest” on page 499.

Managed Code

```
// getters and setters
__property System::String *get_URL();
__property System::String *get_ResourceType();
__property System::String *get_IPaddress();
__property ObAMDateMgd *get_Date();
__property ObAMTimeMgd *get_Time();
__property int get_NumberOfOperations();
__property int get_NumberOfUsers();
   System::String *getOperation(int index);
ObAMIdentityMgd *getUser(int index);
__property bool get_ShowAllowed();
__property bool get_ShowDenied();
__property bool get_ShowMatchingPolicy();
__property bool get_ShowMatchingExpr();
__property void set_URL(System::String *value);
__property void set_ResourceType(System::String *value);
__property void set_AddOperation(System::String *value);
__property void set_IPaddress(System::String *value);
__property void set_Date(ObAMDateMgd *date);
__property void set_Time(ObAMTimeMgd *time);
__property void set_AddUser(ObAMIdentityMgd *value);
__property void set_ShowAllowed(bool value);
__property void set_ShowDenied(bool value);
__property void set_ShowMatchingPolicy(bool value);
__property void set_ShowMatchingExpr(bool value);
```

Class ObAMAccessTestResults

An object of the ObAMAccessTestResults class contains the results of an access test. It includes an array of one or more AccessTestResult objects, described on page 506, one for each user specified in the AccessTest.

Note: The trailing *s* in *Results*

Label	Type	Range	Description
PolicyDomain (no key)	A string Optional	Any	The name of the policy domain that includes the resources specified in the test.
Results	An array of AccessTestResult objects	See the class definition on page 506.	An array of results (AccessTestResult objects), one for each user specified in the test.

Java

```
public class ObAMAccessTestResults {
public String getPolicyDomain();
public int getNumberOfResults();
public ObAMAccessTestResult getResult(int index);
```

C

```
typedef const void * ObAMAccessTestResults_t;
void ObAMAccessTestResults_delete(
                    ObAMAccessTestResults_t results);
const char *ObAMAccessTestResults_getPolicyDomain(
                    ObAMAccessTestResults_t results);
int ObAMAccessTestResults_getNumberOfResults(
                    ObAMAccessTestResults_t results);
ObAMAccessTestResult_t ObAMAccessTestResults_getResult(
                    ObAMAccessTestResults_t results, int index);
```

Obsoleted Methods

Do not use following methods. They have been obsoleted as of NetPoint Access Manager 6.5:

Class ObAMAccessTestResultsMgd

Managed Code. An ObAMAccessTestResultsMgd object contains the results of an access test. See “Class ObAMAccessTestResults” on page 505. It includes an array of one or more AccessTestResultMgd objects, described on page 506, one for each user specified in the AccessTest.

Note: The trailing *s* in *Results*.

```
// getters and setters
__property System::String *get_PolicyDomain();
__property int get_NumberOfResults();
ObAMAccessTestResultMgd *getResult(int index);
```

Class ObAMAccessTestResult

An object of the ObAMAccessTestResult class contains the results of an access test, including the following information:

- The name of the policy, if any, that applies to the resource specified in the test.
- The identity of the user requesting the resource as a test case.
- The name of the authorization expression that applies to the user and the resource.

- The set of determining rules from the expression, if any, that determine the user's access rights and whether access to the resource is authorized for the user.

Label	Type	Range	Description
User (no key)	An Identity Object	See the class definition on page 494.	Information about the user.
Policy	A string Optional	Any	The name of the policy, if any, that applies to the user and the resource.
Authorization Expression	A string, Optional	Any See the class definition on page page 471.	The name of the authorization expression that applies to the user and the requested resource.
Determining Rules	A string, Optional	Any See the class definition on page 464.	The set of one or more rules that contributed to the determination of the outcome of the authorization. These are the rules that determined the result of the authorization expression evaluation.
Authorized	A Boolean flag, Required	0, meaning no, or other, meaning yes.	Answers the question: Is access to the resource authorized for the user?

Note: No trailing *s* in *Result*.

Java

```
public ObAMIdentity getUser();
public String getPolicy();
public boolean getAuthorized();
public String getExpr();
public int getNumberOfDeterminingRules();
public String getDeterminingRule (int index);
public int getAuthorizationStatus();
```

Obsoleted Method

Do not use following method. It has been obsoleted as of NetPoint Access Manager 6.5:

```
public String getRule();
```

Class ObAMAccessTestResultMgd

(Managed Code). An ObAMAccessTestResultMgd object contains the results of an access test for a single user. See “Class ObAMAccessTestResultsMgd” on page 506 and “Class ObAMAccessTestResults” on page 505. For details on the object components, see “Class ObAMAccessTestResult” on page 506.

Note: No trailing *s* in *Result*.

Class ObAMException

The Access Management API Java methods throw exceptions of the class ObAMException when they detect problems with input data or with the connection to the Access Server.

Java

The Java ObAMException class extends the ObAccessException class as follows:

```
public static final int UNDEFINED = 400;
public static final int ADMIN_LOGIN_FAILED = 401;
public static final int NOT_AUTHORIZED = 402;
public static final int BAD_ARGUMENT = 403;
public static final int EXISTING_OBJECT = 404;
public static final int NO_OBJECT = 405;
public static final int BAD_MESSAGE = 406;
public static final int ALREADY_SET = 407;
public static final int FINALIZED = 408;
public static final int UNSUPPORTED_VERSION = 409;
public static final int END_BEFORE_START = 410;
public static final int NO_SET_ADMIN = 411;
public static final int DATA_STORE_ERROR = 412;
public static final int INVALID_LDAP_FILTER = 413;
public static final int MISSING_REQUIRED_PARAM = 414;
public static final int INVALID_PARAM = 415;
public static final int NAME_REQUIRED = 416;
public static final int MODIFY_OBJECT_INVALID = 417;
```

```
public static final int INVALID_PROFILE_ATTRIBUTE = 418;
public static final int AUTHZ_SCHEME_CONFLICT = 419;
public static final int BAD_CHARACTER_DATA = 420;
public static final int CACHE_FLUSH_FAILED = 421;
public static final int AUTHN_SCHEME_PARAM = 422;
public static final int OBJECT_IN_USE = 423;
public static final int CANNOT_DELETE = 424;
public static final int POLICY_RESOURCE_TYPE_MISMATCH = 425;
public static final int INTERNAL_ERROR = 426;
public static final int INVALID_USER = 427;
public static final int INVALID_GROUP = 428;
public static final int FEATURE_NOT_SUPPORTED = 429;
public static final int INVALID_FAILURE_ACTION_ATTIBUTE = 430;
public static final int MISSING_AUTHN_STEP = 431;
public static final int INVALID_AUTHZ_EXPR_SYNTAX = 432;
public static final int AUTHZ_RULE_NOT_FOUND = 433;
public static final int AUTHN_SCHEME_DISABLED = 434;
public static final int INVALID_ACTION_TYPE = 435;
public static final int INVALID_DUPLICATE_ACTIONS_POLICY = 436;
```

Class ObAccessException

C

An ObAccessException object is thrown when the unexpected, unrecoverable problems occur. Because C does not provide an exception mechanism, the C binding includes methods to set up a handler to be called when an exception is thrown. The handler passes the exception, and it can extract the code and data from the exception. For C, the ObAccessExceptionClass is used. For details about the ObAccessException class, see “ObAccessException_t” on page 366.

The following list of ObAccessException codes from the ObAccessException class is used.

```
ObAccessException_AM_UNKNOWN = 400,
ObAccessException_AM_ADMIN_LOGIN_FAILED,
ObAccessException_AM_NOT_AUTHORIZED,
ObAccessException_AM_BAD_ARGUMENT,
ObAccessException_AM_EXISTING_OBJECT,
ObAccessException_AM_NO_OBJECT,
ObAccessException_AM_BAD_MESSAGE,
ObAccessException_AM_GET_OBJECT_IN_SET,
ObAccessException_AM_FINALIZED,
ObAccessException_AM_UNSUPPORTED_VERSION,
ObAccessException_AM_END_BEFORE_START,
ObAccessException_AM_UNSUPPORTED_OPERATION,
ObAccessException_AM_NO_SET_ADMIN,
ObAccessException_AM_DATA_STORE_ERROR,
ObAccessException_AM_READ_DATA_STORE_ERROR,
ObAccessException_AM_INVALID_LDAP_FILTER,
ObAccessException_AM_MISSING_REQUIRED_PARAM,
```

```
ObAccessException_AM_INVALID_PARAM,
ObAccessException_AM_NAME_REQUIRED,
ObAccessException_AM_MODIFY_OBJECT_INVALID,
ObAccessException_AM_INVALID_PROFILE_ATTRIBUTE,
ObAccessException_AM_AUTHZ_SCHEME_CONFLICT,
ObAccessException_AM_BAD_CHARACTER_DATA,
ObAccessException_AM_CACHE_FLUSH_FAILED,
ObAccessException_AM_AUTHN_SCHEME_PARAM,
ObAccessException_AM_OBJECT_IN_USE,
ObAccessException_AM_CANNOT_DELETE,
ObAccessException_AM_POLICY_RESOURCE_TYPE_MISMATCH,
ObAccessException_AM_INTERNAL_ERROR,
ObAccessException_AM_INVALID_USER,
ObAccessException_AM_INVALID_GROUP,
ObAccessException_AM_FEATURE_NOT_SUPPORTED,
ObAccessException_AM_INVALID_FEATURE_ACTION_ATTRIBUTE

typedef void (*ObAccessExceptionHandler2_t)
                          (ObAccessException_t e);
void ObAccessException_setHandler2(
                          ObAccessExceptionHandler2_t handler);
ObAccessExceptionCode_t ObAccessException_getCode(
                          ObAccessException_t e);
const char *ObAccessException_getParameter(
                          ObAccessException_t e, int which);
const char *ObAccessException_toString(ObAccessException_t e);
```

Class ObAccessExceptionMgd

Managed Code

When the Access Management API methods for managed code detect problems, they throw an ObAccessExceptionMgd exception. For enumeration and description of the constants that define the exception codes returned for errors encountered, see “C-Family Status and Error Message Strings” on page 397. For a complete description of the class, see “ObAccessException” on page 351.

```
public:
   ObAccessExceptionMgd: public System::Exception {
   ObAccessExceptionMgd(ObAccessException *ex);
   System::String *getParameter(int index);
   System::String *getCodeString(ObAccessExceptionCode_t code);
   __property System::String *get_String(); };
```

Sample Program

The following listing is part of a sample program that is installed with the NetPoint software in the *NetPoint_install_dir/identity*/oblix/access_server_sdk/samples folder. The sample program uses many of the Access Management API classes. Code listings shown throughout this chapter are excerpted from the sample program.

This portion of the sample program creates a policy domain called My Domain for resources in http://host1/myresources. Here are some of the tasks the code in Listing 43 performs:

- The program sets the default authentication rule for My Domain to use the Basic Over LDAP authentication scheme provided by NetPoint.
- It sets the default audit rule for the policy domain to audit authentication successes, authentication failures, and authorization failures. For these events, it specifies that the user's uid and cn attributes are to be logged.
- It creates an authorization rule that allows J. Smith, J. Himes, and anyone in the group group1 access to the protected resources. On successful authorization, the rule returns a header variable userIs set to the user's uid attribute.
- The application creates a second authorization rule with access conditions which include timing conditions and an IP address requirement. This rule allows anyone access after 12:00:00 of October 15, 2001 on the 1st, 2nd, and 30th of January and November if the day is either a Monday or a Tuesday and the user's browser has an IP address of 192.168.*.*.
- The code adds an action to the second rule that the rule returns on successful authorization.
- The code creates a third authorization rule that allows anyone access to the protected resources.
- The code creates a default authorization expression for the policy domain and it sets the duplicate actions policy to Overwrite. It adds an action to be returned if the authorization expression is evaluated to a result of Inconclusive.
- It sets up two policies within the policy domain.
- It creates an administrator rule to assign delegate administrator rights for the policy domain to J. Smith.
- The code creates the My Domain policy domain. If the policy domain exists, the code throws an exception.
- It gets and displays the names of all policy domains that begin with My. If the code successfully created the new policy domain, the list will include the My Domain policy domain.

Listing 43 Sample Program for Creating a Policy Domain

```
        domain.setName("My Domain");
        domain.setDescription("excerpted from the sample program");
        domain.setEnabled(true);
        ObAMResource resource = new ObAMResource();
        resource.setResourceType("http");
        resource.setHostID("host1");
        resource.setURLPrefix("/myresources");
        domain.addResource(resource);

// Set the default authentication rule

     ObAMAuthenticationRule authnRule = new ObAMAuthenticationRule();
     authnRule.setScheme("NetPoint Basic Over LDAP");
     domain.setDefaultAuthenticationRule(authnRule);

// Set the default audit rule

     ObAMAuditRule auditRule = new ObAMAuditRule();
     auditRule.addEvent(ObAMAuditRule.AUTHENTICATION_SUCCESS);
     auditRule.addEvent(ObAMAuditRule.AUTHENTICATION_FAILURE);
     auditRule.addEvent(ObAMAuditRule.AUTHORIZATION_FAILURE);
     auditRule.addAttribute("uid");
     auditRule.addAttribute("cn");
     domain.setDefaultAuditRule(auditRule);

// Create an authorization rule

     ObAMAuthorizationRule authzRule1 = new ObAMAuthorizationRule();
     authzRule1.setName("Authz Rule 1");
     authzRule1.setEnabled(true);
     ObAMAccessConditions access1 = new ObAMAccessConditions();
     ObAMIdentity person1 = new ObAMIdentity();

     person1.setLoginID("J.Smith");
     access1.addPerson(person1);
     ObAMIdentity person2 = new ObAMIdentity();
     person2.setLoginID("J.Himes");
     access1.addPerson(person2);
     ObAMIdentity group = new ObAMIdentity();
     group.setName("group1");
     access1.addGroup(group);
     person1.setLoginID("admin");
     access1.addPerson(person1);
     authzRule1.setAllowAccessConditions(access1);

// Add a success action to the authorization rule


     ObAMAction action1 = new ObAMAction();
     action1.setType("headerVar");
     action1.setName("UserIs");
```

```
      action1.setValue("uid");action1.setValueType(ObAMAction.ATTRIBUTE);
      authzRule1.addActionOfType(ObAMObjectWithActions.SUCCESS, action1);


// Add the authorization rule to the policy domain

      domain.addAuthorizationRule(authzRule1);


// Create a second authorization rule

      ObAMAuthorizationRule authzRule2 = new ObAMAuthorizationRule();
         authzRule2.setName("Authz Rule 2");
         authzRule2.setEnabled(true);
         authzRule2.setName("Authz Rule 2");
         authzRule2.setEnabled(true);
         ObAMTimingConditions timing2 = new ObAMTimingConditions();
         ObAMDate startDate = new ObAMDate();
         ObAMTime startTime = new ObAMTime();
         startDate.set(2001, ObAMDate.OCTOBER, 31);
         startTime.set(12, 0, 0);
         timing2.setRelativeTo(ObAMTimingConditions.LOCAL_TIME);
         timing2.setStartDate(startDate);
         timing2.setStartTime(startTime);
         timing2.addMonth(ObAMDate.JANUARY);
         timing2.addMonth(ObAMDate.NOVEMBER);
         timing2.addDayOfMonth(1);
         timing2.addDayOfMonth(2);
         timing2.addDayOfMonth(30);
         timing2.addDayOfWeek(ObAMDate.MONDAY);
         timing2.addDayOfWeek(ObAMDate.TUESDAY);
         authzRule2.setTimingConditions(timing2);
         ObAMAccessConditions access2 = new ObAMAccessConditions();
         access2.addIPAddress("192.168.*.*");
         authzRule2.setAllowAccessConditions(access2);

// Add a failure action to the second authorization rule
      ObAMAction action2 = new ObAMAction();
         action2.setType("otherType");
         action2.setName("authzAction");
         action2.setValue("b");
         action2.setValueType(ObAMAction.FIXEDVALUE);
         authzRule2.addActionOfType(
                              ObAMObjectWithActions.FAILURE, action2);

// Add the rule to the domain
         domain.addAuthorizationRule(authzRule2);

// Create a third authorization rule to allow anyone access
```

```
        ObAMAuthorizationRule authzRule3 = new ObAMAuthorizationRule();
        authzRule3.setName("Authz Rule 1");
        authzRule3.setEnabled(true);
        ObAMAccessConditions access3 = new ObAMAccessConditions();
        access3.addRole("Anyone");
        authzRule3.setAllowAccessConditions(access3);

// Add the rule to the domain
        domain.addAuthorizationRule(authzRule3);

// Create a fourth authorization rule to deny anyone access

        ObAMAuthorizationRule authzRule4 = new ObAMAuthorizationRule();
        authzRule4.setName("Authz Rule 2");
        authzRule4.setEnabled(true);
        ObAMAccessConditions access4 = new ObAMAccessConditions();
        access4.addRole("Anyone");
        authzRule4.setDenyAccessConditions(access4);

// Add the rule to the domain

        domain.addAuthorizationRule(authzRule4);

// Create a default authorization expression for the policy domain
        ObAMAuthorizationExpr pd_expr = new ObAMAuthorizationExpr();
        pd_expr.setExpression("Authz Rule 1 & Authz Rule 2");
        pd_expr.setDuplicateActionsPolicy(
                                ObAMAuthorizationExpr.ACTION_OVERWRITE);
// Add an Inconclusive action
        ObAMAction action3 = new ObAMAction();
        action3.setType("otherType");
        action3.setName("authzAction");
        action3.setValue("a");
        action3.setValueType(ObAMAction.FIXEDVALUE);
        pd_expr.addActionOfType(ObAMObjectWithActions.INCONCLUSIVE, action3);

// Add the expression to the policy domain
        domain.setDefaultAuthorizationExpr(pd_expr);

// Set policy 1 for My Domain for GET
// and POST to http://host1/myresources/doc1.html

      ObAMPolicy policy1 = new ObAMPolicy();
      policy1.setName("My Domain Policy 1");
      policy1.setResourceType("http");
      policy1.setHostID("host1");
      policy1.setURLPattern("/myresources/doc1.html");
      policy1.addOperation("GET");
      policy1.addOperation("POST");
      policy1.addResource(resource);
      ObAMAuthorizationExpr p1_authzExpr = new ObAMAuthorizationExpr();
```

```
      p1_authzExpr.setExpression("Authz Rule 1 & Authz Rule 2");
      p1_authzExpr.setDuplicateActionsPolicy(ObAMAuthorizationExpr.UNDEFINED);
      policy1.setAuthorizationExpr(p1_authzExpr);
      domain.addPolicy(policy1);

// Set policy 2 for My Domain for GET
// and POST to http://host1/myresources/prog
// with a query string or post data of progid=1.
      ObAMPolicy policy2 = new ObAMPolicy();
      policy2.setName("My Domain Policy 2");
      policy2.setResourceType("http");
      policy2.setURLPattern("/myresources/prog");
      policy2.addOperation("GET");
      policy2.addOperation("POST");
      policy2.addResource(resource);
      ObAMAuthorizationExpr p2_authzExpr = new ObAMAuthorizationExpr();
      p2_authzExpr.setExpression("Authz Rule 1 & Authz Rule 2");
      p2_authzExpr.setDuplicateActionsPolicy(
                           ObAMAuthorizationExpr.ACTION_IGNORE);
      policy2.setAuthorizationExpr(p2_authzExpr);
      ObAMParameter parameter2 = new ObAMParameter();
      parameter2.setName("progid");
      parameter2.setValue("1");
      policy2.addParameter(parameter2);
      domain.addPolicy(policy2);
// Set an admin rule for My Domain that gives delegate
// rights to J.Smith

   ObAMAdminRule adminRule = new ObAMAdminRule();
   ObAMIdentity adminPerson = new ObAMIdentity();
   adminPerson.setLoginID("J.Smith");

   adminPerson.setLoginID("admin");
   adminRule.addPerson(adminPerson);
   domain.setDelegateAdminRule(adminRule);

// Create My Domain. If it already exists, this will throw an exception.

   am.setPolicyDomain(domain, ObAccessManager.CREATE);
// Get and display all domains that start with My
      ObAMPolicyDomain[] domains =
                  am.getPolicyDomains(ObAccessManager.MAX,
      "My", ObAccessManager.BEGINS_WITH);
         if (domains != null) {
            for (int i = 0; i < domains.length; i++) {
                     displayPolicyDomain(0, "Policy Domain:", domains[i]);
            }
         };
      }
```

7 Authentication Plug-in API

The NetPoint Access Server uses both authentication and authorization controls to limit access to resources that it protects. Authentication is governed by authentication rules. The authentication rules use authenticating schemes, and the schemes use one or more *plug-ins* to do tests on the credentials provided by a user when he or she tries to access a resource. The plug-ins can taken from a set provided as part of the Access Server installation, or they can be custom plug-ins.

This chapter provides information on creating custom authentication plug-ins. It covers the following topics:

About the Authentication Plug-in API

When a browser requests a resource from a NetPoint-protected Web server, the Access Server checks to see if the resource is protected, and if it is, whether the user needs to authenticate. If the user has not already logged in, the Access Server requires a new login for the user and sends an authentication challenge to the browser. The challenge conforms to the challenge method defined in an *Authentication Scheme*. The authentication scheme is part of an *Authentication Rule* which is part of the access policy protecting the resource. When the scheme is carried out, it invokes an authentication plug-in, or two or more chained plug-ins that are performed in the order specified in the scheme.

All schemes follow the same general flow. In response to an authentication challenge, the browser obtains credentials from the user, such as a user name and password or a client certificate. In some cases, for example client certificate authentication, the browser generates credentials on behalf of the user. The browser sends the credentials to the server, in a format determined by the challenge. The Access Server re-formats the credentials as a set of name:value pairs for use during processing, and treats them as an authentication request.

The user's credentials are input to the single plug-in or to each plug-in in the scheme. Output is a status to accept, continue, deny or abort the authentication, plus a set of credentials, possibly different from the originals.

If authentication fails, result messages are logged in the Access Server audit file, if it is provided by the plug-in. When the authentication scheme finishes, the result *must* have produced one and only one valid user DN, or no user DN.

If authentication succeeds, the Access Server creates a session cookie containing the user's profile DN, the IP address of the user's browser, the level of the authentication scheme, and the expiration time for the cookie. The Access Server can also set HTTP header variables based on the authentication actions defined for the authentication scheme. The cookie and HTTP information are returned to the browser, and access is granted.

If authentication does not succeed, the Access Server sets an HTTP return status of 401 (Not Authorized), the standard response for unauthenticated access, and access to the resource is denied.

Authentication schemes can be created which use only the predefined NetPoint Authentication plug-ins; see "NetPoint Standard Plug-Ins" on page 558. In addition, developers can create their own plug-ins and use them in schemes, alone or in combination with the standard plug-ins.

C API Environment

The following sections describe the development environment for the plug-in API, including support file location and major components.

Support Files Location for the C API

When you install the Access Server component of the Access System, you put it into an installation directory, *ASInstall_Dir,* for example:

C:/NetPoint/access/oblix

Sample files for the authentication plug-in API are installed within this directory, at:

AccessServer_install_dir/access/oblix/sdk/authentication/samples

The samples directory contains two subdirectories, valicert and authn_api. The valicert directory contains the source code used to create the Valicert plug-in provided with the Access System.

The samples directory does not contain all of the files required to build and run the source code. For example, the header files, include files, and others are not provided. Valicert licensing requirements prohibit Oblix from including the complete set of files required to compile the sample source code. To compile and run this sample plug-in, you must obtain a license for the Valicert SDK.

Authn_api contains source code for a simple example authentication plug-in, an example makefile and, one level down in the include directory, the header file authn_api.h.

The file authn_api.h contains two important sets of information:

- Definitions of the set of utilities that the Access Server provides to all authentication plug-ins
- Definitions of the API data and functions

Note: Some of the definitions provided in this file are essential in order to correctly build and operate the API. When the plug-in is loaded by the Access Server, it expects to find the set of five functions in authn_api.h implemented within the plug-in. You may add information to the file, but *do not remove any of the existing content*.

To build

1. Under the samples directory, copy the content of the authn_api directory to a second directory, for example myplugin.
2. Within the new directory, change the content of the authn.c file, and/or create additional files to provide the desired functionality specific to your plug-in.
3. Change the makefile to show the actual path to your C compiler and to the authn_api.h file, and to include and compile all of your source code.

Run the makefile. (The example supplied applies only to UNIX; you need to create your own for the Windows environment.)

The resulting *.so or *.dll is your new plug-in.

C API Plug-in Directory

The plug-in you created *must* be stored in the system where the Access Server is running, in the directory:

ASInstall_Dir`/lib`

The Access Administrator needs to know the filename of the plug-in, and its required and optional data names in order to properly configure the plug-in into an authentication scheme. See the *NetPoint 7.0 Administration Guide Volume 2* for a more detailed discussion of authentication scheme configuration. Following the “C Authentication Plug-in Example” on page 540 is a screen showing the authentication scheme configuration used to support the example.

C API Data

This section describes the various types of constant and variable data that the API uses.

Defines (C)

Authn_api.h predefines several values for use as fixed argument values when working with some functions.

```
#define OB_AN_PLUGIN_VERSION"5.0"
   #define ObAnPluginRequestResource  "Resource"
   #define ObAnPluginRequestOperation "Operation"
   #define ObAnPluginRequesterDN      "RequesterDn"
   #define ObAnPluginRequesterIP      "RequesterIP"
```

Value Name	Meaning
OB_AN_PLUGIN_VERSION	When ObAnPluginGetVersion is called, this value *must* be returned to the Access Server. The value in the header file may change with releases of the product.
ObAnPluginRequestResource	The resource after the host name and port, for example: /basic/page.htm.
ObAnPluginRequestOperation	The operation being performed on the resource.
ObAnPluginRequesterDN	If an authentication plug-in has set the DN, this is where other plug-ins can access that DN. The plug-in named credential_mapping always sets the DN. Custom authentication plug-ins can set the DN, once they have determined what it is, by calling SetCredFn.
ObAnPluginRequesterIP	The IP address of the client that issued this request.

Handles (C)

The Access Server and API use pointers, also called *handles,* to allow manipulation of data structures that the Access Server maintains for use by the plug-in. These handles are named and described below. The description of content for all structures begins at “C Structures” on page 525. The terms *list*, *name*, *value* and *item* describe the data relationships within the ObAnPluginInfo structure, described on 526.

Data Type/Name	Purpose
`void const* ObAnPluginSVData_t`	A handle to a data structure containing names that are singlevalued. It is used to locate Creds and ActionInfo in the ObAnPluginInfo structure.
`void* ObAnPluginMVData_t`	A handle to a data structure containing names that can be multivalued. They are used to locate Params and Context in the ObAnPluginInfo structure.
`char** ObAnPluginStatusMsg_t`	A NULL-terminated string that a plug-in function returns to report on the result of the function.
`void const* ObAnPluginList_t`	A handle pointing to a list of items for a multivalued name that is part of the data for the Params or Context members of the structure ObAnPluginInfo. This handle is obtained using the function GetData.

Data Type/Name	Purpose
`void const* ObAnPluginListItem_t`	A handle pointing to the current item in a list for a multivalued name. This handle is obtained using the functions GetFirstItem or GetNext.
`struct ObAnPluginInfo* ObAnPluginInfo_t`	A handle pointing to the structure ObAnPluginInfo that contains information from the Access Server and data generated by the plug-in.
`struct ObAnServerContext* ObAnServerContext_t`	A handle pointing to the structure ObAnServerContext. The structure provides general information about the Access Server.

C Return Values

Many of the functions the Access Server and API use to communicate are expected to return a status value. These are all predefined in several categories as described here.

ObAnActionType_t

These are action flags that the API returns to the Access Server, to tell it what actions to take.

```
typedef enum {
   ObAnSuccessRedirect = 0,
   ObAnFailRedirect = 1,
   ObAnSuccessProfileAttrs = 2,
   ObAnFailProfileAttrs = 3,
   ObAnSuccessFixedVals = 4,
   ObAnFailFixedVals = 5
} ObAnActionType_t;
```

ObAnSuccessRedirect	For successful authentication, tells the Access Server to set the redirection URL, as defined in the authentication rule.
ObAnFailRedirect	For failed authentication, tells the Access Server to set the redirection URL, as defined in the authentication rule.
ObAnSuccessProfileAttrs	For successful authentication, tells the Access Server to set the profile attributes for the action using values provided by the authentication rule, which may have been added to by the plug-in.
ObAnFailProfileAttrs	For failed authentication, tells the Access Server to set the profile attributes for the action using values provided by the authentication rule, which may have been added to by the plug-in.
ObAnSuccessFixedVals	For successful authentication, tells the Access Server to set the fixed values defined in the authentication rule, which may have been added to by the plug-in.
ObAnFailFixedVals	For failed authentication, tells the Access Server to set the fixed values defined in the authentication rule, which may have been added to by the plug-in.

ObAnPluginstatus_t

Plug-ins must return one of these values to the Access Server to show the result of the attempt to authenticate.

```
typedef enum {
     ObAnPluginstatusContinue = 0,
     ObAnPluginstatusAllowed = 1,
     ObAnPluginstatusDenied = 2,
     ObAnPluginstatusAbort = 3
}ObAnPluginstatus_t;
```

Value Name	Meaning
`ObAnPluginStatusContinue`	Your plug-in returns this code if it completed execution successfully. The Access Server interprets a return code of Continue as indication that the current plug-in succeeded, and that it should continue processing the plug-ins of the steps of the authentication scheme. If all of the plug-ins of an authentication scheme return this result code, authentication is successful.
`ObAnPluginStatusAllowed`	Your plug-in returns this code if authentication is successful. The Access Server interprets a return code of Allowed as indication that credentials were processed and authentication succeeded. The Access Server performs no further processing of authentication plug-ins in any of the steps of the authentication scheme.
`ObAnPluginStatusDenied`	Your plug-in returns this code if authentication failed. The Access Server interprets a return code of Denied as indication that credentials were processed and authentication failed. The Access Server performs no further processing of authentication plug-ins of any steps of the authentication scheme because authentication failed.
`ObAnPluginStatusAbort`	Your plug-in returns this code if a fatal error occurs during its processing. The Access Server interprets a return code of Abort as direction to terminate the entire authentication process specified by the authentication scheme, not just the step containing the plug-in. If Abort is returned during initialization, the Access Server will log the condition, but it will not terminate the process.

ObAnASStatus_t

When a plug-in calls upon the Access Server to perform one of its built-in functions, the Access Server attempts to execute the function, and returns one of these values.

```
typedef enum {
   ObAnASStatusSuccess = 0,
   ObAnASStatusFailed
}ObAnASStatus_t;
```

Value Name	Meaning
ObAnASStatusSuccess	The Access Server successfully performed the operation.
ObAnASStatusFailed	The Access Server did not perform the operation. The most likely cause of this error is that the plug-in tried to change values that it is not allowed to change. An example is an attempt to add a second value for a name, to an array member which allows only single values.

C Structures

The Access Server groups related data items into named structures, allocates the memory for them, and carries the data for them. Structures are opaque to the developer, meaning that they can be used to transfer information to and from the Access Server, but the developer cannot change the way the structure is organized or the format of the data that it contains. The content of the structure can be changed in some instances.

Here are the structures used in the Authentication Plug-in API.

ObAnServerContext

This structure carries information about the Access Server that the plug-in may need.

```
struct ObAnServerContext {
   char    *AccessServerInstallDir;
   char    *AccessServerAnPluginAPIVersion;
   };
```

ObAnServerContext_t is the handle that works with this structure.

Data held in this structure is read only.

Data Type/Name	Purpose
AccessServerInstallDir	Path to the Installation directory for the NetPoint Access Server, for example NetPoint/Access. Note: this does not include the /oblix directory.
AccessServerAnPlugin APIVersion	The lowest authentication plug-in API version the instance of the Access Server supports.

ObAnPluginInfo

The Access Server fills this structure with data determined by the Authentication Rule to be used, and provides the filled structure to the plug-in. The plug-in modifies data within the structure and may append new data to it as work progresses through the plug-in. When there are multiple plug-ins being carried out as part of an authentication scheme, it also provides a means to set variable information for the first plug-in in the rule, and pass information from one plug-in to another within the scheme.

```
struct ObAnPluginInfo {
   ObAnPluginsVData_t Creds;
   ObAnPluginMVData_t Params;
   ObAnPluginMVData_t Context;
   ObAnPluginsVData_t ActionInfo;
   };
```

ObAnPluginInfo_t is the handle pointing to this structure.

Data is extracted from structures and stored to them using the functions described under “Functions Provided by the Access Server (C API)” on page 529.

Data Type/Name	Purpose
`Creds`	Creds is all information submitted by the entity (user or application) trying to access a resource. The plug-in may add to or replace this data. The data is passed to the next plug-in in sequence and hence can be used by the plug-in to communicate with the plug-in following it in an authentication scheme. The Access Server provides four predefined names within this list: Resource, Operation, RequesterDN, and RequesterIP.
`Params`	The parameters specified in the plug-in configuration. The plug-in may add to or replace this data, within the plug-in. However, this data is *not* passed to the next plug-in.
`Context`	Data created by the plug-in. The plug-in may add to or replace this data. The data is passed to the next plug-in in sequence.
`ActionInfo`	Action information specified in the plug-in configuration. The plug-in may add to or replace this data. Actions can also be set and passed to the next plug-in, keeping in mind that the final user of the action information is the Access Server.

Understanding the organization of the ObAnPluginInfo structure in is key to understanding how the Authentication Plug-in API works. This structure can be thought of as an array of four nested arrays, the structure *members*. Each member holds one or more *names*.

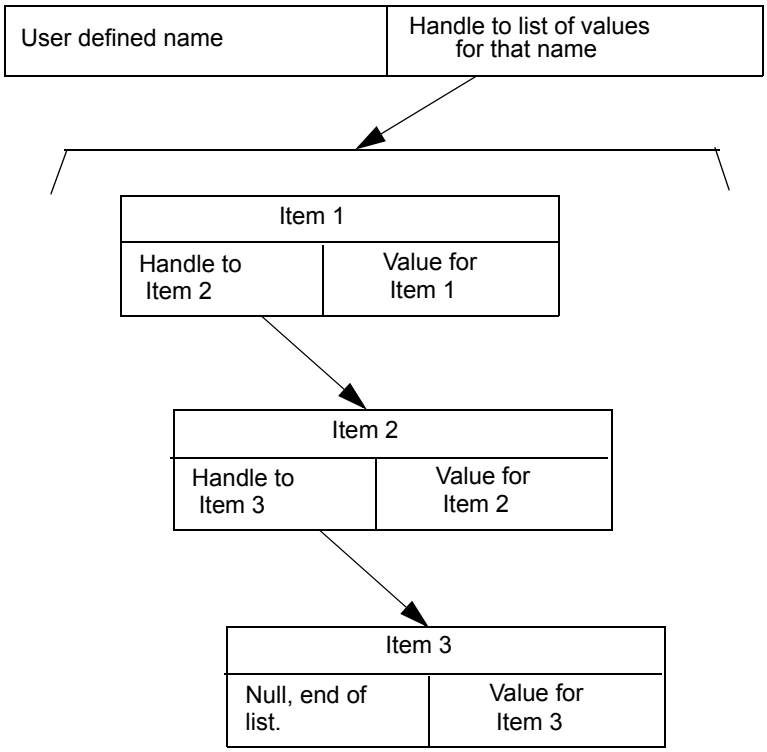
For structure members that are of type ObAnPluginSVData_t, the array consists of a set of names and the single value associated with each one. For example, the creds member can be thought of this way:

"Resource"	1 value
"Operation"	1 value
"RequesterDN"	1 value
"RequesterIP"	1 value
User Created Credential Name	1 value

To get the value, you use GetCredFn and provide the name of the credential for which you want the value. The function returns the single value. GetActionFn operates the same way, with actions.

For structure members that are of type ObAnPluginMVData_t, each name has an associated handle that points to a list of one or more items. Each item contains a value and a handle to the next item in the list. A handle for the next item set to NULL indicates the end of the list.

The params member can be thought of according to the diagram below. You use GetDataFn to get the pointer to the list, for a specified parameter name within the params array.You then use GetFirstFn to get a handle to the information for the first item in the list. GetValueFn at this point returns the value for item 1, GetNextFn returns the handle to the information for item 2, and so on.



ObAnPluginFns

This structure provides handles to a block of functions available in the Access Server, which the plug-in uses to manipulate data in the ObAnPluginInfo structure.

```
struct ObAnPluginFns {
   ObASPluginGetFirstItem_tGetFirstItemFn;
   ObASPluginGetValue_t    GetValueFn;
   ObASPluginGetNext_t     GetNextFn;
   ObAnPluginGetData_t     GetDataFn;
   ObAnPluginsetData_t     SetDataFn;
   ObAnPluginGetCred_t     GetCredFn;
   ObAnPluginsetCred_t     SetCredFn;
   ObAnPluginGetAction_t   GetActionFn;
   ObAnPluginSetAction_t   SetActionFn;
   ObAnPluginSetAuthnUid_t SetAuthnUidFn
   };
```

ObAnPluginFns_t is the handle pointing to this structure.

You refer to this structure when implementing ObAnPluginFn. The functions are described separately under “Functions Provided by the Access Server (C API)” on page 529.

C API Functions

Functions used by the API to talk to the Access Server are of two types. They can be provided by the Access Server, in which case they are called by reference to it. Otherwise, they must be implemented in the plug-in, following the prototypes in authn_api.h.

Functions Provided by the Access Server (C API)

To use these functions you must call them out as members of the structure of type ObAnPluginFns that you named in your code. For example, if you implemented ObAnPluginFn and set the variable name of type ObAnPluginFns to pFnBlock, then you call GetCredFn by reference to its place in the structure, as pFnBlock->GetCredFn.

GetDataFn

This function returns a handle to a list of multivalued data for either the Params or the Context members of the ObAnPluginInfo structure. Given the handle, the plug-in must then use the list manipulation functions—GetFirstItemFns, GetValueFn, GetNextFn, GetValueFn and so on—to extract information from the list.

The function takes the form:

```
ObAnPluginList_t GetDataFn(
   ObAnPluginMVData_t prequesterinfo,
   const char*        pName
   );
```

Input Parameters:

Variable Name	Purpose
`requesterInfo`	Handle to a multivalued member of the ObAnPluginInfo structure passed to the plug-in.
`pName`	Retrieves the list of items for this name.

Output Parameters

There are no output parameters for this function.

The function returns a handle to a list of values for the given name. If the handle value is NULL, the name is not present for the structure member.

SetDataFn

This function sets a value for a name in the Params or Context members of the ObAnPluginInfo structure. The Access Server checks to see if the name already exists, and appends or overwrites the value, depending on the value of the replace flag.

The function takes the form:

```
ObAnASStatus_t SetDataFn(
   ObAnPluginMVData_t pRequesterContext,
   const char*        pName,
   const char*        pValue,
   const int          replace
   );
```

Input Parameters

Variable Name	Purpose
`pRequesterContext`	Handle to a writable, multivalued member of the ObAnPluginInfo structure passed to the plug-in.
`pName`	Name for the information whose value is to be set.
`pValue`	The value to be inserted.
`replace`	Specifies whether to replace or append to existing values for the name. An integer value of 0 indicates append, all other values are a request to replace the current first value for pName.

Output Parameters

There are no output parameters from this function.

The function returns one of the values for ObAnASStatus_t.

Note: The replace option applies only to the first item in the list.

GetFirstItemFn

This function returns the handle to the first item in a list of multivalued data, once a handle to the full list has been obtained using GetDataFn. You must do at least this first step before asking for a value using GetValueFn. The handle to the full list is *not* the handle to the first item in the list.

The function takes the form:

```
ObAnPluginListItem_t GetFirstItemFn(
   ObAnPluginList_t   plist
   );
```

Input Parameters

Variable Name	Purpose
pList	The handle to the full list of values.

Output Parameters

There are no output parameters for this function.

The function returns a handle to the first item in a list of values. If the handle value is NULL, there is no first item.

GetValueFn

This function gets a value from an item in a multivalued list, once a handle to the item has been obtained, using GetFirstItemFn or GetNextFn.

The function takes the form:

```
const char* GetValueFn(
   ObAnPluginListItem_t  PItem
   );
```

Input Parameters

Variable Name	Purpose
pItem	Handle to an item.

Output Parameters

There are no output parameters for this function.

The function returns the string value of an item.

GetNextFn

This function gets a handle to the next item in a multivalued list, given the handle to the current item, which you get either with GetFirstItemFn or an earlier use of GetNextFn.

The function takes the form:

```
ObAnPluginListItem_t GetNextFn(
   ObAnPluginListItem_t  pItem
   );
```

Input Parameters

Variable Name	Purpose
pItem	Handle to the current item in the list.

Output Parameters

There are no output parameters for this function.

The function returns a handle to the next item in the list. If the value returned is NULL, then you have reached the end of the list.

GetCredFn

This function retrieves the value corresponding to a credential name, given the handle to the credential information and the name. A user's authentication level can be tied to a scheme or to a resource where the user has logged in. For example, you can set the authentication level based on whether the user has logged in through Active Directory or a reverse proxy. The custom plug-in extracts this information by querying the Access Server for an ObAuthentSchemeLevel variable that is maintained in a credential list.

Only one value is allowed per credential name.

The function takes the form:

```
const char *GetCredFn(
   ObAnPluginSVData_t pCreds,
   const char*        pName
   );
```

Input Parameters

Variable Name	Purpose
pCreds	Handle for the credential information passed to the plug-in.
pName	Name of the credential information to retrieve.

Output Parameters

There are no output parameters for this function.

The function returns the value for the credential, or NULL if no value is found.

Example

```
schemeLevel = pFnBlock->GetCredFn(pInfo->Creds, "ObAuthentSchemeLevel");
```

SetCredFn

This function sets a value for a specified name in the credential information passed to the plug-in, given the handle to the credential information, the name of the credential to modify, and the value to store. If the name already exists, its current value is overwritten. If not, the name and value are added.

The function takes the form:

```
ObAnASStatus_t SetCredFn(
   ObAnPluginSVData_t pCreds,
   const char*        pName,
   const char*        pValue
   );
```

Input Parameters

Variable Name	Purpose
pCreds	Handle for the credential information passed to the plug-in.
pName	Key/name for the credential information to set.
pValue	The value to write to the credentials

Output Parameters

There are no output parameters for this function.

The function returns one of the values for ObAnASStatus_t.

Example

```
schemeLevel = pFnBlock->GetCredFn(pInfo->Creds, "ObAuthentSchemeLevel");
if(schemeLevel != NULL){
schemeLevelAsInt = atoi(schemeLevel);
schemeLevelAsInt +=10;
itoa(schemeLevelAsInt, buff, 10);
pFnBlock->SetCredFn(pInfo->Creds, "ObAuthentSchemeLevel", buff);
```

Example: To set the UID

This function sets the uid that is internal to authentication for the current user. If the uid has already been set, its value is overwritten. If not the value is added. The following example illustrates setting the UID for the user:

```
   pFnBlock->SetCredFn(pInfo->Creds, ObAnPluginRequesterDN, "cn=Halley Starks,
ou=LHuman Resource, ou=Los Angles, ou=Dealer1k1, ou=Latin America, ou=Ford,
o=Company,c=US");
```

GetActionFn

This function retrieves action information, given the handle to the action information, the name of the action to retrieve, and its type. Only one value is allowed per action name/type combination.

The function takes the form:

```
const char *AnGetAction(
   ObAnPluginSVData_t pActionInfo,
   const char*        pName,
   ObAnActionType_t   pActionType);
```

Input Parameters

Variable Name	Purpose
pActionInfo	Handle to action information passed to the plug-in.
pName	Key/name for the action information to retrieve.
pActionType	Type of action information to get.

Output Parameters

There are no output parameters for this function.

The function returns the value for the action/type combination, or NULL if no value is found.

SetActionFn

This function sets a value in the action information passed to the plug-in, using the handle to the action information, the name of the action to modify, the value to store, and the action type. If the name and type combination already exists, its value is overwritten. If not, the name and value are added.

The function takes the form:

```
ObAnASStatus_t SetActionFn(
   ObAnPluginSVData_t pCreds,
   const char*        pName,
   const char*        pValue
   ObAnActionType     pActionType);
```

Input Parameters

Variable Name	Purpose
pActions	Handle for the action information passed to the plug-in.
pName	Key/name for the action information to set.
pValue	The value to write to the action.
pActionType	Type of action information to set.

Output Parameters

There are no output parameters for this function.

The function returns one of the values for ObAnASStatus_t.

SetAuthnUidFn

This function sets the uid that is internal to authentication for the current user. If the uid has already been set, its value is overwritten. If not the value is added.

Important: This function is obsolete. Use SetCredFn instead.

The function takes the form:

```
ObAnASStatus_t SetAuthnUidFn(
   char* pUid
   );
```

Input Parameters

Variable Name	Purpose
pUid	The new uid to be set.

Output Parameters

There are no output parameters for this function.

The function returns one of the values for ObAnASStatus_t.

C Functions Implemented in the Plug-in

All of the functions described in this section *must be* implemented by the developer in the plug-in, following the prototype provided in authn_api.h. The Access Server expects them all to be present and refuses to execute the plug-in if they are not.

The OBDLLEXPORT entry for each function is *required*. It provides a means for the Access Server to locate and call these functions from within the plug-in.

The Access Server calls the functions in this order:

- **GetVersion**—The first time the plug-in is loaded.
- **Init**—The first time the plug-in is loaded.
- **DeAllocStatusMessage**—Automatically, following any of the other functions which returns a status message.
- **Fn**—Each time the plug-in is used.
- **Terminate**—When the Access Server shuts down, or the plug-in is unloaded.

ObAnPluginGetVersion

The Access Server calls this function immediately after the plug-in is loaded. The plug-in returns the version number of the library with which it was built. The Access Server uses this version to determine if it can support the plug-in. That is, it would catch a situation in which an older version of the Access Server was being asked to support a newer version of the API, or a newer version of the Access Server was being asked to work with a plug-in version no longer supported.

The function takes the form:

```
OBDLLEXPORT
const char* ObAnPluginGetVersion(void);
```

Input parameters

There are no input parameters to this function.

Output parameters

There are no output parameters from this function.

The function returns the version of the authentication plug-in. This is the value set by the following line in the authn_api.h file:

```
#define OB_AN_PLUGIN_VERSION"X.X"
```

ObAnPluginInit

The Access Server calls this function immediately after making the version check. The function initializes the workspace for the plug-in, which could include tasks such as connecting to a database and initializing global data for the plug-in. This function, and the two others where a presult string might be built, must allocate memory in order to return the presult string. The Access Server automatically calls the ObAnPluginDeallocStatusMsg function to deallocate this memory.

The function takes the form:

```
OBDLLEXPORT
   ObAnPluginStatus_t ObAnPluginInit(
   ObAnServerContext_t   pServerContext,
   ObAnPluginStatusMsg_t pResult
   );
```

Input Parameters

Variable	Purpose
pServerContext	Context information of the Access Server.

Output Parameters

Variable	Purpose
pResult	Result message reported by the function.

The function should return one of the ObAnPluginStatus_t values, either ObAnPluginStatusContinue or ObAnPluginStatusAbort.

ObAnPluginTerminate

The Access Server calls this function when it is about to terminate. The function should be written to clear whatever workspace the plug-in set for itself and disconnect from any other applications, such as a database, that the plug-in might have opened.

The function takes the form:

```
OBDLLEXPORT
   ObAnPluginStatus_t ObAnPluginTerminate(
   ObAnServerContext_t   pServerContext,
   ObAnPluginstatusMsg_t pResult);
```

Input Parameters

Variable	Purpose
pServerContext	Context information from the Access Server.

Output Parameters

Variable	Purpose
pResult	Result message reported by the function.

The function should return one of two ObAnPluginStatus_t values, either ObAnPluginStatusContinue or ObAnPluginStatusAbort.

ObAnPluginFn

The Access Server calls this function to request the plug-in to do the actual custom authentication work using the plug-in information and the server context. This function can also modify some of the plug-in information for use by subsequent authentication plug-ins in a scheme.

The function takes the form:

```
OBDLLEXPORT
   ObAnPluginStatus_t ObAnPluginFn(
   ObAnServerContext_t  pServerContext,
   ObAnPluginFns_t      pFuncBlock,
   ObAnPluginInfo_t     pData
   ObAnPluginstatusMsg  pResult
   );
```

Input Parameters

Variable	Purpose
pServerContext	Context information for the Access Server.
pFuncBlock	Handle to a block of functions provided by the Access Server, that the plug-in needs to manipulate data.
pData	Handle to data passed to the plug-in.

Output Parameters

Variable	Purpose
pData	Handle to data modified by the plug-in.
pResult	Result message reported by the function.

The function can return any one of the four ObAnPluginstatus_t values.

ObAnPluginDeallocStatusMsg

If another of these implemented functions sent a pResult string, the Access Server automatically calls this function to delete the memory that was assigned by the plug-in to build the message. You should ensure that it does so.

The function takes the form:

```
OBDLLEXPORT
      void ObAnPluginDeallocStatusMsg(
         ObAnPluginstatusMsg_t pStatusMsg
         );
```

Input Parameters

Variable	Purpose
pStatusMsg	Status Message to be deallocated.

Output Parameters

There are no output parameters from this function.

The function returns nothing

C Authentication Plug-in Example

Listing 44 shows some basic uses of the plug-in functions.

Listing 44 Authentication Plug-in Example.

```
/*
* Copyright (c) 1996-2001, Oblix Inc. All Rights Reserved.
*
* authn_api.c

* Custom Authentication plugin.
*
* This implementation of the 5 authentication functions is built
* into a DLL named SimpleAPI, and then, in a slightly modified
* form, into a DLL called SimpleAPI2.
* The differences between the two forms of the plugin are these:
* 1) the second form writes the number 2 into all of its
*    log messages
* 2) the second form does not make a check on the param
*    values or write context data back to the info structure.
*
* It shows:
*
* 1. Example implementations of all 5 functions.
* 2. Examples for extracting data from many of the structures.
* 3. An example of writing new data to the info structure.
* 3. A simple way to log results for testing.
*/

#include "authn_api.h"
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <iostream.h>
#include <malloc.h>
#include <fstream.h>

#ifdef _WIN32
#undef strcasecmp
#define strcasecmp stricmp
#endif

/*
* Implementation of ObAnPluginGetVersion
*
* The data logged by this function appears only once, when the
* Plugin is first loaded.
*
*/
```

Listing 44 (Continued) Authentication Plug-in Example.

```
OBDLLEXPORT const char* ObAnPluginGetVersion(void)
{
   FILE *file = fopen("d:\\ANtestfile.txt", "a+");
   fprintf (file, "\n%s%s\n", "sending Authn version,
      which is ",OB_AN_PLUGIN_VERSION);
   fclose(file);
   return OB_AN_PLUGIN_VERSION;
}
/*
* Implementation of ObAnPluginInit
*
* The logged data appears only once, when the Plugin is first loaded.
*
*/
OBDLLEXPORT ObAnPluginStatus_t ObAnPluginInit(
      ObAnServerContext_t pContext,
      ObAnpluginStatusMsg_t pResult)
{
// Values to be read in by this function are initialized.
   ObAnpluginStatus_t rtval;
   const char* pASPluginVersion = NULL;
   const char* pASInstallDir    = NULL;
   FILE *file = fopen("d:\\ANtestfile.txt", "a+");
   fprintf (file, "\n%s\n", "initializing");
   if(pContext != NULL) {
      pASPluginVersion = pContext->AccessServerAnPluginAPIVersion;
      pASInstallDir    = pContext->AccessServerInstallDir;
      fprintf (file, "\n%s%s\n", "  version is ",pASPluginVersion);
      fprintf (file, "\n%s%s\n", "  AS directory is ",
      pASInstallDir);
   }
   if((pASPluginVersion != NULL) &&
       (strcmp(pASPluginVersion, OB_AN_PLUGIN_VERSION) == 0)) {
      rtval = ObAnPluginStatusContinue;
   *pResult = strdup("Success version check");
   }
   else {
/*
* return failure, because the version provided by the AS
* is not what was expected.
*/
      rtval = ObAnPluginStatusAbort;
   }
   fclose(file);
   return rtval;
}
```

Listing 44 (Continued) Authentication Plug-in Example.

```
/*
* Implementation of ObAnPluginTerminate
* The logged data appears only when the Access Server terminates.
*/
OBDLLEXPORT ObAnPluginStatus_t ObAnPluginTerminate(
      ObAnServerContext_t pContext,
      ObAnPluginStatusMsg_t pResult)
{
   FILE *file = fopen("d:\\ANtestfile.txt", "a+");
   fprintf (file, "\n%s\n", "terminating gracefully");
   *pResult = strdup("Success, terminated");
    fclose(file);
   return ObAnPluginStatusContinue;
}
/*
* Implementation of ObAnPluginDeallocStatusMsg
* The logged data appears following each other function
* that provides a presult.
*/
OBDLLEXPORT void ObAnPluginDeallocStatusMsg(
      ObAnPluginStatusMsg_t pResult)
{
   FILE *file = fopen("d:\\ANtestfile.txt", "a+");
   fprintf (file, "\n%s\n", "deallocating");

   if(pResult != NULL && *pResult != NULL) {
      free(*pResult);
      *pResult = NULL;
   }
    fclose(file);
}
/*
* Implementation of ObAnPluginFn
*/
OBDLLEXPORT ObAnpluginstatus_t ObAnPluginFn(
      ObAnServerContext_t pContext,
      ObAnPluginFns_t pFnBlock,
      ObAnPluginInfo_t pInfo,
      ObAnPluginStatusMsg_t pResult)
{
// rtval is initialized to allow continuing to the next
// plugin
   ObAnPluginStatus_t rtval = ObAnPluginStatusContinue;

   ObAnPluginList_t list;
   ObAnPluginListItem_t item;
   ObAnASStatus_t writeres=100;
```

Listing 44 (Continued) Authentication Plug-in Example.

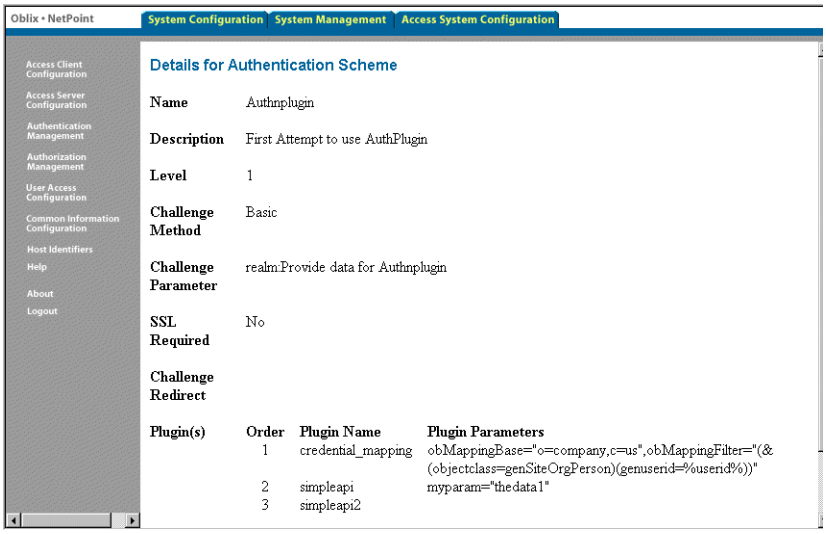
```
// These are initialized, and get overwritten with real data
// if any is found below
   const char* Resource;
   const char* Operation;
   const char* UserDN;
   const char* con1   = "cdummy1";
   const char* con2   = "cdummy2";
   const char* con3   = "cdummy3";
   const char* param1 = "pdummy1";
   /* this initialization is key to the example*/
   const char* param2 = NULL;
   const char* param3 = "pdummy3";
   FILE *file = fopen("d:\\testfilen_fn.txt", "a+");
   fprintf (file, "\n%s\n", "Starting Fn");
// This is an example of getting data from the Creds member
// of Info, using the predefined names
   Resource =pFnBlock->GetCredFn
         (pInfo->Creds,ObAnPluginRequestResource);
   Operation=pFnBlock->GetCredFn
         (pInfo->Creds,ObAnPluginRequestOperation);
   UserDN   =pFnBlock->GetCredFn
         (pInfo->Creds,ObAnPluginRequesterDN);
   fprintf (file, "\n%s%s\n", "resource is ",Resource);
   fprintf (file, "\n%s%s\n", "operation is ",Operation);
   fprintf (file, "\n%s%s\n", "user DN   is ",UserDN);
// This set of code tries to extract the context information.
//
   list   = pFnBlock->GetDataFn(pInfo->Context,"isthereany");
// For the first use of the plugin, none has been set.
// There is no context data, and no context name called isthereany.
//
// For the second use of the plugin, context data may be present
// and if so will be read in.
   if (list != NULL){
      fprintf(file, "\n%s\n","found context data!!");
      item   = pFnBlock->GetFirstItemFn(list);
      if (item != NULL){
         con1 = pFnBlock->GetValueFn(item);
         item   = pFnBlock->GetNextFn(item);
         if (item != NULL){
            con2 = pFnBlock->GetValueFn(item);
            item   = pFnBlock->GetNextFn(item);
            if (item != NULL){
               param3 = pFnBlock->GetValueFn(item);
            }
         }
      }
   }
```

Listing 44 (Continued) Authentication Plug-in Example.

```
fprintf (file, "\n%s%s\n", "first context   is ",con1);
   fprintf (file, "\n%s%s\n", "second context  is ",con2);
   fprintf (file, "\n%s%s\n", "third context   is ",con3);
// This set of code extracts the param information.
// The value of param2, which is set, or not, in the Authentication
// Scheme, controls the test.
   list   = pFnBlock->GetDataFn(pInfo->Params,"myparam");
   if (list != NULL){
      item   = pFnBlock->GetFirstItemFn(list);
      if (item != NULL){
         param1 = pFnBlock->GetValueFn(item);
         item   = pFnBlock->GetNextFn(item);
         if (item != NULL){
            param2 = pFnBlock->GetValueFn(item);
            item   = pFnBlock->GetNextFn(item);
            if  (item != NULL){
               param3 = pFnBlock->GetValueFn(item);
            }
         }
      }
   }

   fprintf (file, "\n%s%s\n", "first param   is ",param1);
   fprintf (file, "\n%s%s\n", "second param  is ",param2);
   fprintf (file, "\n%s%s\n", "third param   is ",param3);
   *pResult = strdup("Success");
// -----> The second form of the plugin omits the code from here
// If there is a value set for param2 under the myparam
// name in the authentication scheme, then the logic here
// returns allowed to the Access Server, and the next plugin
// will not be used.
// If there is no value set for param2, then the logic here
// sets a value in the context data, which will be seen by the
// next plugin.
   if (param2 != NULL){
      rtval = ObAnPluginStatusAllowed;
      fprintf (file, "\n%s\n", "second param is not NULL");
   }
   else {
      fprintf (file, "\n%s\n", "second param was NULL");
      writeres=pFnBlock->SetDataFn
            (pInfo->Context, "isthereany", "context1", 0);
      fprintf (file, "\n%s%i\n", "AS returned", writeres);
   }
// -----> to here
   fclose(file);
   return rtval;
}
```

The following screen shows the setup for the Authentication Scheme that uses the example plug-in. Note that no second value is entered for the myparam parameter name.



Given this version of the authentication scheme, the corresponding trace information from the example code is:

```
sending Authn version, which is 5.0
initializing
version is 5.0
AS directory is
d:\netscape\server4\ws41sp6\docs\techpubs_fcs88\as\access

deallocating
in 2, sending Authn version, which is 5.0

in 2, initializing
version is 5.0

AS directory is
d:\netscape\server4\ws41sp6\docs\techpubs_fcs88\as\access

Starting Fn
resource  is /test3
operation is GET
user DN   is cn=Rohit Valiveti,ou=Sales,ou=Dealer1k1,ou=Latin
America,ou=Ford,o=Company,c=US
first context   is cdummy1
second context  is cdummy2
third context   is cdummy3
first param   is thedata1
second param  is (null)
third param   is pdummy3
```

```
second param was NULL

AS returned 0

in 2, Starting Fn
resource  is /test3
operation is GET
user DN   is cn=Rohit Valiveti,ou=Sales,ou=Dealer1k1,ou=Latin
America,ou=Ford,o=Company,c=US
found context data!!
first context   is context1
second context  is cdummy2
third context   is cdummy3
first param   is pdummy1
second param  is (null)
third param   is pdummy3
```

Managed Code API Environment

The following sections describe the development environment for the managed plug-in API, including support file location and major components.

Support Files Location for the Managed Code API

When you install the Access Server component of the Access System, you put it into an installation directory, *ASInstall_Dir,* for example:

C:/NetPoint/access/oblix

Sample files for the authentication plug-in API are installed within this directory, at:

ASInstall_Dir/sdk/managed/authn++

Authn_api contains source code for a simple example authentication plug-in, an example project file. Some of the definitions in this file are essential for building and operating the API. When the plug-in is loaded by the Access Server, it expects to find the set of four functions in authn_api.h implemented within the plug-in.

To build

1. Under the samples directory, copy the content of the authn_api directory to a second directory, for example myplugin.
2. Within the new directory, change the content of the authn.c file, and/or create additional files to provide the desired functionality specific to your plug-in.
3. Modiy the project to include and compile all of your source code.

4. Build the project.

   The resulting *.dll is your new plug-in.

5. See “Managed Code API Plug-in Directory” on page 547 for details about storing the plug-in and implementing an example.

Managed Code API Plug-in Directory

The plug-in you created *must* be stored in the system where the Access Server is running, in the directory:

ASInstall_Dir/lib

The Access Administrator needs to know the filename of the plug-in, and its required and optional data names in order to properly configure the plug-in into an authentication scheme. See the *NetPoint 7.0 Administration Guide Volume 2* for a more detailed discussion of authentication scheme configuration.

Following the “C Authentication Plug-in Example” on page 540 is a screen showing the authentication scheme configuration used to support the example. To use the example with managed code, be sure to add the following in the parameters text box for the authn_api plug-in:

obtype=“managed”, obnamespace=“sample”

Managed Code API Data

This section describes the various types of constant and variable data that the API uses.

Defines (Managed Code)

The managed_plugin_interface.h predefines several values for use as fixed argument values when working with some functions.

```
pubic_value class AnDefines {
   static string *ObAnPluginResourceRequest= S"Resource";
   static string *ObAnPluginRequestOperation= S"Operation";
   static string *ObAnPluginRequesterIP = S"RequesterIP";
   static string *ObAnPluginRequesterDN    = S"RequesterDN";
   static string *OB_AN_PLUGIN_VERSION = S"5.0";
};
```

Value Name	Meaning
OB_AN_PLUGIN_VERSION	When ObAnPluginGetVersion is called, this value *must* be returned to the Access Server. The value in the header file may change with releases of the product.
ObAnPluginRequestResource	The resource after the host name and port, for example: /basic/page.htm.
ObAnPluginRequestOperation	The operation being performed on the resource.
ObAnPluginRequesterDN	If an authentication plug-in has set the DN, this is where other plug-ins can access that DN. The plug-in named credential_mapping always sets the DN. Custom authentication plug-ins can set the DN, once they have determined what it is, by calling SetCredFn.
ObAnPluginRequesterIP	The IP address of the client that issued this request.

Interfaces (Managed Code)

The Access Server and API use interfaces to allow manipulation of data structures that the Access Server maintains for use by the plug-in. These interfaces are named and described below.

Interface	Purpose
IObAnPluginSVData	An interface that provides functions to get singe-valued data. It is used to locate Creds and ActionInfo in the ObAnPluginInfo interface.
IObAnPluginMVData	An interface that provide functions to multivaled data. They are used to locate Params and Context in the ObAnPluginInfo interface.
IObASPluginListItem	An interface pointing to an item in a list.
IObAnPluginInfo	An interface providing functions that contain information from the Access Server and data generated by the plug-in.
IObAnServerContext	An inteface that provides functions that contain general information about the Access Server.

The Access Server groups related data items into named structures, allocates the memory for them, and carries the data for them. Structures are opaque to the developer, meaning that they can be used to transfer information to and from the Access Server, but you cannot change the way the structure is organized or the format of the data that it contains. The content of the structure can be changed in some instances.

The following are the interfaces that are used in the Authentication Plug-in API.

IObAnServerContext

This interface provides information about the Access Server that the plug-ins may need.

```
public _gc _interface IObAnServerContext {
   _property String* get_AccessServerInstallDir();
   _property String* get_AccessServerAnPluginAPIVersion();
};
```

Data Type/Name	Purpose
`get_AccessServerInstallDir`	Path to the installation directory for the NetPoint Access Server, for example, NetPoint/Access. Note: this does not include the /oblix directory.
`get_AcessServerAnPluginAPIVersion`	The lowest authentication plug-in API version that the instance of the Access Server supports.

IObAnPluginInfo

The Access Server creates an object with data determined by an authentication rule, and provides the interface to the plug-in. The plug-in modifies data within the structure and may append new data to it as work progresses through the plug-in. When there are multiple plug-ins being carried out as part of an authentication scheme, it also provides a means to set variable information for the first plug-in in the rule, and passes information from one plug-in to another within the scheme.

```
public _gc _interface IObAnPluginInfo {
   IObAnPluginSVData*   get_Creds();
   IObAnPluginMVData*   get_Params();
   IObAnPluginMVData*   get_Context();
   IObAnPluginSVData*   get_ActionInfo();
```

```
};
```

Function	Purpose
`get_Creds`	The creds function consists of all information submitted by the user or application that is trying to access a resource. The plug-in may add to or replace this data. The data is passed to the next plug-in in sequence. As a result, the data can be used by the plug-in to communicate with the plug-in following it in an authentication scheme. The Access Server provides four predefined names within this list: Resource, Operation, RequesterDN, and RequesterIP.
`get_Params`	The parameters specified in the plug-in configuration. The plug-in may add to or replace this data within the plug-in. This data is not passed to the next plug-in.
`get_Context`	Data created by the plug-in. The plug-in may add to or replace this data. The data is passed to the next plug-in in sequence.
`get_ActionInfo`	Action information that is specified in the plug-in configuration. The plug-in may add to or replace this data. Actions can also be set and passed to the next plug-in, but the final user of the action information is the Access Server.

This interface can be thought of as an array of four nested arrays, the structure members. Each member holds one or more names.

IObAnPluginSVData

To get the value, you use the GetCred function in the IObAnPluginSVData interface, and provide the name of the credential for which you want the value. The function returns a single value. GetAction operates in the same with with actions.

For structure members that are of type IObAnPluginSVData, the array consists of a set of names and a single value associated with each one. For example, the creds member can be though of this way:

`"Resource"`	`1 value`
`"Operation"`	`1 value`
`"RequesterDN"`	`1 value`
`"RequesterIP"`	`1 value`
`User Created Credential Name`	`1 value`

Function	Purpose
`GetCred`	This function retrieves the value corresponding to a credential name, given the handle to the credential information and the name. A user's authentication level can be tied to a scheme or to a resource where the user has logged in. For example, you can set the authentication level based on whether the user has logged in through Active Directory or a reverse proxy. The custom plug-in extracts the infomration by querying the Access Server for an ObAuthntSchemeLevel variable that is maintained in a credential list.
`SetCred`	This function sets a value for a specified name in the credential information passed to the plug-in, given the handle to the credential information, the name of the credential to notify, and the value to store. If the name already exists, its current value is overwritten. If not, the name and value are added.
`GetAction`	This function retrieves action information, given the hndle to the action information, the name of the action to retrieve, and its type. Only one value is allowed per action name/type combination.
`SetAction`	This function sets a value in the action information passed to the plug-in, using the handle to the action information, the name of the action to modify, the value to store, and the action type. If the name and type combination already exists, its value if overwritten. If not, the name and value are added.

IObAnPluginMVData

The IObAnPluginMVData interface provides functions to work on objects that contain a list of one or more items. Each item contains a value.

Function	Purpose
`get_Data`	This function returns a handle to a list of multivalued data for either the Params or the Context members of the IObAnPluginInfo interface. Given the handle, the plug-in must then use the list manipulation functions—GetFirstItemFns, GetValueFn, GetValueFn, and so on—to extract information from the list.
`set_Data`	This function sets a value for a specified name in the credential information passed to the plug-in, given the handle to the credential information, the name of the credential to modify, and the value to store. If the name already exists, its current value is overwritten. If not, the name and value are added.

IObAsPluginListItem

The list returned by the get_Data function can be traversed using standard Enumerator functions. Each item in the list is returned whose value can be obtained by using this interface.

Function	Purpose
`get_Value`	This interface provides the function to get the value of an item.

Managed Code Return Values

Many of the functions the Access Server and API use to communicate are expected to return a status value. These are all predefined in several categories as described here.

ObAnActionType

These are action flags that the API returns to the Access Server, to tell it what actions to take.

```
_value enum ActionType {
   ObAnSuccessRedirect = 0,
   ObAnFailRedirect = 1,
   ObAnSuccessProfileAttrs = 2,
   ObAnFailProfileAttrs = 3,
   ObAnSuccessFixedVals = 4,
   ObAnFailFixedVals = 5
};
```

ObAnSuccessRedirect	For successful authentication, tells the Access Server to set the redirection URL, as defined in the authentication rule.
ObAnFailRedirect	For failed authentication, tells the Access Server to set the redirection URL, as defined in the authentication rule.
ObAnSuccessProfileAttrs	For successful authentication, tells the Access Server to set the profile attributes for the action using values provided by the authentication rule, which may have been added to by the plug-in.
ObAnFailProfileAttrs	For failed authentication, tells the Access Server to set the profile attributes for the action using values provided by the authentication rule, which may have been added to by the plug-in.
ObAnSuccessFixedVals	For successful authentication, tells the Access Server to set the fixed values defined in the authentication rule, which may have been added to by the plug-in.
ObAnFailFixedVals	For failed authentication, tells the Access Server to set the fixed values defined in the authentication rule, which may have been added to by the plug-in.

ObAnPluginstatus

Plug-ins must return one of these values to the Access Server to show the result of the attempt to authenticate.

```
_value enum ObAnPluginStatus {
      ObAnPluginstatusContinue = 0,
      ObAnPluginstatusAllowed = 1,
      ObAnPluginstatusDenied = 2,
      ObAnPluginstatusAbort = 3
};
```

Value Name	Meaning
ObAnPluginStatusAbort	You set this value to show that a fatal error occurred while processing authentication. Processing for this authentication request does not continue after this plug-in.
ObAnPluginStatusAllowed	Credentials were processed and authentication succeeded. No further authentication plug-ins are processed.
ObAnPluginStatusDenied	Credentials were processed and authentication failed. Processing does not continue after the function. Authentication fails.
ObAnPluginStatusContinue	Authentication processing continues after the function. If all plug-ins in an authentication scheme return continue, authentication is implicitly allowed.

ObAnASStatus

When a plug-in calls upon the Access Server to perform one of its built-in functions, the Access Server attempts to execute the function, and returns one of these values.

```
_value enum ASStatus {
   ObAnASStatusSuccess = 0,
   ObAnASStatusFailed
};
```

Value Name	Meaning
ObAnASStatusSuccess	The Access Server successfully performed the operation.
ObAnASStatusFailed	The Access Server did not perform the operation. The most likely cause of this error is that the plug-in tried to change values that it is not allowed to change. An example is an attempt to add a second value for a name, to an array member which allows only single values.

Managed Code Functions Implemented in the Plug-in

All of the functions described in this section *must be* implemented by the developer in the plug-in, following the prototype provided in managed_plugin_interface.h.

For Authentication plug-ins, you must define a class with the following functions:

```
public _gc class ObAuthzPlugin
{
```

```
                public:
                     ObAuthnPlugin();
                     String* ObAnPluginGetVersion();
                     IObAuthnPlugin::Status ObAnPluginInit
(Oblix::IObAnServerContext* context, String* msg);
                     IObAuthnPlugin::Status
ObAnPluginFN(Oblix::IObAnServerContext* context, Oblix::IObAnPluginInfo* info);
                     IObAuthnPlugin::Status ObAnPluginTerminate
(Oblix::IObAnServerContext* context, String* msg);
                };
```

The Access Server calls the functions in this order:

- **ObAnPluginGetVersion**—The first time the plug-in is loaded.
- **ObAnPluginInit**—The first time the plug-in is loaded.
- **ObAnPluginFn**—Each time the plug-in is used.
- **ObAnPluginTerminate**—When the Access Server shuts down, or the plug-in is unloaded.

ObAnPluginGetVersion

The Access Server calls this function immediately after the plug-in is loaded. The plug-in returns the version number of the library with which it was built. The Access Server uses this version to determine if it can support the plug-in. That is, it would catch a situation in which an older version of the Access Server was being asked to support a newer version of the API, or a newer version of the Access Server was being asked to work with a plug-in version no longer supported.

The function takes the form:

```
OBDLLEXPORT
const char* ObAnPluginGetVersion(void);
```

Input parameters

There are no input parameters to this function.

Output parameters

There are no output parameters from this function.

The function returns the version of the authentication plug-in. This is the value set by the following line in the authn_api.h file:

```
#define OB_AN_PLUGIN_VERSION"X.X"
```

ObAnPluginInit

The Access Server calls this function immediately after making the version check. The function initializes the workspace for the plug-in, which could include tasks such as connecting to a database and initializing global data for the plug-in. This function, and the two others where a presult string might be built, must allocate memory in order to return the presult string. The Access Server automatically calls the ObAnPluginDeallocStatusMsg function to deallocate this memory.

The function takes the form:

```
OBDLLEXPORT
   ObAnPluginStatus_t ObAnPluginInit(
   ObAnServerContext_t   pServerContext,
   ObAnPluginStatusMsg_t pResult
   );
```

Input Parameters

Variable	Purpose
`pServerContext`	Context information of the Access Server.

Output Parameters

Variable	Purpose
`pResult`	Result message reported by the function.

The function should return one of the ObAnPluginStatus_t values, either ObAnPluginStatusContinue or ObAnPluginStatusAbort.

ObAnPluginTerminate

The Access Server calls this function when it is about to terminate. The function should be written to clear whatever workspace the plug-in set for itself and disconnect from any other applications, such as a database, that the plug-in might have opened.

Input Parameters

Variable	Purpose
`pServerContext`	Context information from the Access Server.

Output Parameters

Variable	Purpose
`pResult`	Result message reported by the function.

The function should return one of two ObAnPluginStatus_t values, either ObAnPluginStatusContinue or ObAnPluginStatusAbort.

ObAnPluginFn

The Access Server calls this function to request the plug-in to do the actual custom authentication work using the plug-in information and the server context. This function can also modify some of the plug-in information for use by subsequent authentication plug-ins in a scheme.

Input Parameters

Variable	Purpose
`pServerContext`	Context information for the Access Server.
`pFuncBlock`	Handle to a block of functions provided by the Access Server, that the plug-in needs to manipulate data.
`pData`	Handle to data passed to the plug-in.

Output Parameters

Variable	Purpose
`pData`	Handle to data modified by the plug-in.
`pResult`	Result message reported by the function.

The function can return any one of the four ObAnPluginstatus_t values.

Troubleshooting

For unit testing of plug-ins, writing the results to a file as the example here does is the best approach. The pResult text is captured by the Access Server audit log only if authentication fails, and then only for Solaris. If you write to a file, be sure you have the correct permissions for writing into the directory holding the file.

Performance is a developer responsibility and should be considered when designing a plug-in. The total time required to process one authentication request depends on the performance of all the plug-ins that are invoked while processing that request.

Plug-ins are trusted by the Access Server. No access check is performed when giving pre-configured information to the plug-in.

A coding problem in a plug-in, such as a memory or access violation or segmentation, or bus error fault, can crash the Access Server.

Plug-ins allow optional parameters, which would usually be filled in by an Access Administrator when schemes are created. Plug-ins should be able to gracefully handle the situation in which values for these parameters are not supplied.

If requests seem to fail without reason, check the path of the shared library to be sure it is correct; it *must* be available at *ASInstall_Dir*`/lib`. Also check to be sure that the Authentication Scheme refers to the correctly spelled shared library name.

Be sure that the ObAuthnplug-inInit function gets called and does not return abort (turn auditing on for authentication failure and see if anything appears in the audit logs.)

NetPoint Standard Plug-Ins

NetPoint provides several Authentication plug-ins as part of a standard installation. They are described in this section. You can use these plug-ins in combination with your own custom plug-ins to create Authentication Schemes.

NetPoint supports several challenge methods (Basic, X.509 Certificate, and Form), as described in the *NetPoint 7.0 Administration Guide Volume 2*. The plug-ins described here can be used with challenge methods as follows:

- Credential Mapping—All methods
- Validate Password—Basic, Form
- Certificate Decode—X.509 Certificate
- Selection Filter—All methods

- ValiCert—X.509 Certificate
- NT/Windows 2000—Basic, Form
- Secure ID—Form

Credential Mapping Plug-In

Name	`credential_mapping`	
Purpose	Maps user-entered information to match a valid Distinguished Name (DN) in the directory	
Result	If one, and only one, DN matches the specified criteria, execution of the authentication scheme continues. The obMappingBase and obMappingFilter are added to the list of credentials and the uid carried in the credentials is set to the value of the DN. Authentication fails if zero or more than one DN is returned.	
Dependency	This plug-in *should* be present in the scheme, if you are going to need a DN.	
Parameters	`obMappingBase`	Base DN for the LDAP search. If omitted or empty, the directory base is used.
	`obMappingFilter`	Filter for the LDAP search. This parameter is required.
	`ObDomain`	Used only when authentication is being done against a specific directory in an Active Directory Forest and the authentication method is basic. Value for this parameter is one of the configured Directory Server profile names.
	`obEnableCredentialCache`	Turns off the credential mapping cache in the credential_mapping plugin. Using this parameter, a deactivation takes effect the next time the user authenticates. If you deactivate a user while they are logged in, the user will still have access to resources based on policy information. However, if the obEnableCredentialCache parameter is set to false, once the user's session token expires or the user logs out, the next time the user is authenticated, they are not allowed into a protected site.
Example Parameters	`obMappingBase="o=Company, c=US", obMappingFilter="(&(objectclass=inetOrgPerson)(uid=%userid%))"`	

By default, the credential mapping cache is turned on. The following table shows the values for obEnableCredentialCache:

obEnableCredentialCache

Value	Interpretation
no value	Credential mapping cache turned on
true	Credential mapping cache turned on
false	Credential mapping cache turned off

An example of the credential_mapping Authentication plug-in with credential mapping cache turned off:

```
        credential_mapping
obMappingBase="%domain%",obMappingFilter="(&(&(objectclass=user)(samaccountname=
%userid%))(|(!(obuseraccountcontrol=*))(obuseraccountcontrol=ACTIVATED)))",obdom
ain="domain",obEnableCredentialCache="false"
```

To set the obEnableCredentialCache parameter

1. In the System Console, select Access System Configuration.
2. Click Authentication Schemes.
3. Select the Authentication Scheme you want to modify.
4. Click Modify.

Add the obEnableCredentialCache="false" parameter to the credential_mapping plug-in.

Note: For details about context-specific data for an authentication request and form-based authentication to pass the originally requested URL to a change password servlet, see the *NetPoint 7.0 Administration Guide Volume 2*.

Validate Password Plug-In

Name	`validate_password`	
Purpose	Validates the password entered at the browser against the user's password in the directory.	
Result	If the user-entered password matches the password in that user's directory entry, execution of the authentication scheme continues. If not, it fails. Nothing is added to the list of credentials.	
Dependency	This plug-in requires a valid Distinguished Name (DN) therefore it is best to call the credential_mapping plug-in before calling this plug-in.	
Parameters	`obCredentialPassword`	Specifies the name of the password field. This parameter must be listed first.
	`obAnonUser`	(optional) Specifies a userid that is considered authenticated with any password, for example, guest or anonymous. This userid must map to a user profile in the directory, preferably one with restricted access. Multiple obAnonUser parameter values are allowed for a single plug-in.
Example Parameters	`obCredentialPassword="password"` `obAnonUser="cn=anonymous, o=Company, c=US"`	

Certificate Decode Plug-In

Name	`cert_decode`
Purpose	Decodes an X.509 certificate and extracts the components of the certificate's subject and issuer DNs.
Result	If the decoding of the certificate is successful, for each component the plug-in inserts a credential with a certSubject or certIssuer prefix. For instance, if your certificates have a subject name such as givenName=somename, the plug-in will add the credential certSubject.givenName=somename to the credential list. If not, authentication fails.
Dependency	The browser must supply an X.509 certificate as part of the credentials.
Parameters	None
Notes	For additional information on this plug-in, see the *NetPoint 7.0 Administration Guide Volume 2*.

Selection Filter Plug-In

Name	`selection_filter`	
Purpose	Applies a filter to a user's credentials.	
Result	If the credentials meet the criteria specified by the filter, the credentials are accepted and authentication continues. If not, authentication fails.	
Dependency	The data to be substituted in ObSelectionfilter must be in credentials before this plug-in runs. For the example parameters provided below, cert-decode must run before selection-filter, in order to provide the certissuer information.	
Parameters	`obSelectionFilter`	Filter to apply to the user credentials.
Example Parameters	`obSelectionFilter="(%certIssuer.CN%=Verisign Class I CA)"`	

ValiCert Plug-In

Name	`authn_valicert`	
Purpose	Accesses ValiCert's Validation Authority server with the necessary certificates to validate the ValiCert client certificate.	
Result	If ValiCert returns a successful validation, authentication continues. If not, authentication fails.	
Dependency	The browser must provide the certificate as part of the credentials.	
Parameters	`vaURL`	The URL to the ValiCert Validation Authority (VA).
Example Parameters	`vaURL="http://ocsp.valicert.net"`	

NT/Win2000 Plug-In

Name	`authn_windows`
Purpose	Given a username, password, and domain in the credentials, authenticates these against an NT domain, Windows 2000 domain, or Windows 2000 Kerberos. When using the Basic challenge method, the domain should be entered in the username field. However, when using the Forms challenge method you may want to provide a field in the form for the user to type in a distinct domain name.

Result	If authn_windows returns success, authentication continues. If not, authentication fails.	
Dependency	When using the Basic Challenge method, the domain name should be entered in the username field in the form Domain\Name. When using the Forms Challenge method, the form can be designed to include a field specifically for domain.	
Parameters	`ntusername`	Name of the parameter containing the username.
	`ntpwd`	Name of the parameter containing the password.
	`ntdomain`	(optional) Name of the parameter containing the domain.

Example Parameters	`ntusername="userid"`
	`ntpwd="password"`
	`ntdomain="domain"`	

SecurID Plug-In

Name	`authn_securid`			
Purpose	Authenticates the credentials from a user's SecurID token against a SecurID ACE Server. The plug-in supports the SecurID Next Tokencode Mode and New PIN Mode operations.			
Result	If the ACE Server returns success, authentication continues. If not, authentication fails.			
Dependency	None			
Parameters	Name	Default	REQ/OPT	**Comments**
	`fullformdir`	none	REQ	This is the full path to the location of the SecurID forms.
	`machine`	none	REQ	This is the name of the Web server actually doing the SecurID authentication. If you are redirecting all SecurID authentications, this is the Web server name that you are redirecting to. For example: securid.abc.com:8888.
	`username`	`login`	OPT	If you are using the sample forms, set this value to login. If you want to specify a value, you must provide its value, as a creds challenge parameter for the plug-in, and also in the ObMappingFilter parameter for the Credential Mapping plug-in.
	`passcode`	`password`	OPT	If you are using the sample forms, set this value to password. If you want to specify a different value, you must provide it as a creds challenge parameter for the plug-in.

Parameters	`formdir`	*<WebGate_install_dir>*/securid-forms/	OPT	This is the path, relative to where the WebGate is installed, to the SecurID forms. This requires a trailing slash(/). If you want to use a different path, you use the new value here and for fullformdir. You must also change the value for the form challenge parameter when configuring the plug-in.
	`httpType`	http://	OPT	This is the type of Web server that is handling SecurID authentications. The two valid values are http:// and https://.
	`choiceLabel`	choice	OPT	This is the name of the field in the HTML form corresponding to the user's choice of how a PIN is generated. Set this using the creds challenge parameter.
	`newpinLabel`	newpin	OPT	This is the name of the field in the HTML form corresponding to the new PIN entered by the user. Set this using the creds challenge parameter.
	`newpinLabel2`	newpin2	OPT	This is the name of the field in the HTML form corresponding to the user's re-entered new PIN. Set this using the creds challenge parameter.
Example Parameters	`fullformdir="<`*WebGate_install_dir*`>/access/oblix/securid-forms/" machine="securid.abc.com:8888"`			

8 Authorization Plug-in API

The NetPoint Access Server controls access to resources by requiring requestors to be both *authenticated* and *authorized*. The chapter "Building AccessGates with the Access Server SDK" on page 245 describes how the Access Server works. "Authentication Plug-in API" on page 517 describes support for authentication, the process by which users establish and prove their identities in order to gain access.

This chapter describes support for authorization. Authorization is the determination of what types of operations users are permitted to perform after they have been authenticated.

This chapter discusses the following topics:

About the Authorization Plug-In API

The Authorization plug-in API provides a way for developers to create modules, called plug-ins, that are used within an authorization *scheme*. Schemes are included in authorization rules, and one or more authorization rules, along with one authentication rule and one audit rule, make up a *policy* that controls access to resources within a *domain*, such as URLs within a Web site or a set of methods within an application. NetPoint provides two standard resource types, HTTP and EJB, but others can be easily added and defined by administrators. See the *NetPoint 7.0 Administration Guide Volume 2* for instructions on creating resource types, domains, policies, rules and schemes.

Plug-ins within authorization schemes are used for two purposes:

- To confirm or deny access to a resource, or to acquire data to be used by the next authorization rule in the policy. This is called an *authorization plug-in*.
- To perform an action of some sort after the access decision is made. This is called a *custom action plug-in*.

Execution of an authorization plug-in delivers one of the following results, which are described in more detail in the sections that follow, at “ObAzplug-instatus_t” on page 574.

- Continue
- Access allowed
- Access denied
- Abort

Custom action plug-ins can be directed to execute if the authorization plug-in has allowed access or if access has been denied. They are used to return data to the AccessGate or to perform a service, such as to notify some person or log a transaction.

To use a plug-in created by the Authorization Plug-in API, two types of information need to be configured by an administrator.

- An authorization scheme to use the plug-in. A given scheme can use both authorization plug-ins and custom action plug-ins.
- A custom authorization rule to use the scheme.

To create the plug-in itself, refer to the other sections in this chapter:

- “API Environment” on page 569, to find out where the API library is installed, its build environment, and how to use it.
- “C API Data” on page 571, to understand the data used within the API.

- "C API Functions" on page 581, to understand what the API does.
- "C Example" on page 590, to see the API in use.
- "Troubleshooting" on page 607, to avoid some possible problem areas.

Support for C and Managed Code

Historically, NetPoint has supported writing authorization plug-ins in C. As of NetPoint 6.5, you can now write these plug-ins using any language supported by the Microsoft .NET framework, including C, MC++, and Visual Basic. If you are using the plug-in in a Windows environment, managed code enables you to select from a variety of implementation languages and provides the other benefits of managed code.

API Environment

C Code Location

The authorization plug-in SDK is installed as a standard component when the Access Server is installed, at

ASInstall_dir`/sdk/authorization/samples`

where *ASInstall_dir* is the location where you have installed the Access Server, for example

```
NetPoint/access/oblix
```

The samples directory contains an example of plug-in code, and one or more make files, as well as an include subdirectory. The include subdirectory contains two header files, to be included in the plug-ins to be written. The file as_plugin_utils.h defines a set of utilities that the Access Server provides to all authorization plug-ins. authz_plugin_api.h defines the API data and functions, and includes the other header file.

Note: The header file contains definitions for the API data and functions. Content provided for this file as part of the NetPoint installation is essential in order to correctly build and operate the API. When the plug-in is loaded by the Access Server, it expects to find the set of functions implemented in authz_plugin_api.h available within the plug-in. You may add information to the file, but *do not remove any of the existing content*.

To build

1. Under the samples directory, create a new directory named, for example, myplugin. Copy the make files and sample code to this new directory.

   Within the new directory, the authz_api.c file provides a good example of the structure and operation of a plug-in. You probably want to create your own file to add some functionality specific to your site.

2. Change the make file to show the actual path to your C compiler, and to include and compile all of your source code.

3. Run the make file.

   The resulting *.so or *.dll is your new plug-in.

Managed C++ Code Location

The authorization plug-in SDK is installed as a standard component when the Access Server is installed, at

`ASInstall_dir/sdk/authorization/managed/authz_c++`

where *ASInstall_dir* is the location where you have installed the Access Server, for example,

`NetPoint/access/oblix`

The directory contains an example of plug-in code in C++. The file managed_plugin_interface in the following location:

`ASInstall_dir/apps/common/bin`

defines a set of interfaces that the Access Server provides to all managed authorization plug-ins. Managed_plugin_interface.h defines and documents the interfaces that can be used by the plug-in writer.

Note that the header file contains definitions for the API data and functions. Content provided for this file as part of the NetPoint installation is essential in order to correctly build and operate the API. When the plug-in is loaded by the Access Server, it expects to find the set of functions implemented in managed_plugin_interface.h available within the plug-in.

Important: Do not remove any of the existing content.

To build

1. Under the samples directory, create a new directory.

   This directory can have any name, for example, mydirectory.

2. In the new directory, the file cplusplus.cpp provides an example of the structure and operation of a plug-in.

   You will probably want to create a new file and add functionality specific to your site.

3. Use the cplusplus.vcproj project file to load and build the plug-in.

   The resulting .dll is your new plug-in.

Plug-in Location

The plug-in you create (in C or in managed code), as either a *.so or *.dll file, can be stored anywhere on the system where the Access Server is running. To be consistent with Authentication Plug-in APIs, you can copy Authorization plug-ins to:

`<$`*ASInstall_dir*`>/lib`

This is not a requirement, however; the file can be stored anywhere on the machine running the Access Server. For this reason, the NetPoint Access System Administrator needs to know the full path to the file, to be able to refer to the plug-in when configuring an authorization scheme. The Administrator also needs to know the required and optional input parameters needed by the plug-in. See the *NetPoint 7.0 Administration Guide Volume 2* for a full discussion of authorization scheme configuration. Following the "C Example" on page 590 is a screen showing the authorization scheme configuration used to support the example.

Note: NetPoint's migration tools do not automatically carry forward custom plug-ins. This is another good reason to put all of these in one place, to make the manual migration task easier.

C API Data

C Constant Definitions

The authz_plugin_api.h file includes several defined values to aid in programming.

One provides the value that is returned to the Access Server when ObAzPluginGetVersion is called:

```
#define OB_AZ_PLUGIN_VERSION "4.6"
```

Note: The value provided for the version may differ for later versions of NetPoint.

Others map to names for data content that is provided by the Access Server in either the RequestorInfo or RequestContext members of the ObAzPluginInfo structure:

```
#define ObAzPluginRequesterDn "RequesterDn"
#define ObAzPluginRequesterIP "RequesterIP"
#define ObAzPluginRequestResourceType "ResourceType"
#define ObAzPluginRequestResource "Resource"
#define ObAzPluginRequestOperation "Operation"
```

C Handles

The Access Server and API use pointers, also called *handles,* to allow manipulation of data structures that the Access Server maintains for use by the plug-in. These handles are named and described below. The description of content for all structures begins at “C Structures” on page 576. The terms *list*, *name*, *value* and *item* describe the data relationships within the ObAzPluginInfo structure, described on “ObAzPluginInfo” on page 576.

Data Type/Name	Purpose
void const* ObASPluginList_t	A handle pointing to a list of values for a named member of the structure ObAzPluginInfo. This handle is obtained using the function GetDataFn.
void const* ObASPluginListItem_t	A handle pointing to one of the items within a list of values. This handle is obtained using GetFirstItemFn or GetNextFn.
ObASPluginListItem_t *ObASPluginGetFirstItem_t	A handle pointing to an Access Server function that gets the handle to the head of a list for a name.
const char* *ObASPluginGetValue_t	A handle pointing to an Access Server function that gets the value for the current item.
ObASPluginListItem_t *ObASPluginGetNext_t	A handle pointing to an Access Server function that gets the handle to the next item in a list.
ObASPluginList_t *ObAzPluginGetData_t	A handle pointing to an Access Server function that gets the handle to the head of a list for a name.
ObAzASStatus_t *ObAzplug-insetData_t	A handle pointing to an Access Server function that stores a value to a list.
void const* ObAzPluginData_t	A handle to the head of the list of names for any of the read-only members of the opaque data structure, ObAzPluginInfo.
struct ObAzPluginFns ObAzPluginFns_t	A handle pointing to the structure `ObAzPluginFns` that contains handles to functions provided by the Access Server, used to manipulate data in the ObAzPluginInfo structure.
`struct ObAzPluginInfo*` `ObAzPluginInfo_t`	A handle pointing to the head of the structure ObAzPluginInfo.
`char**` `ObAzplug-instatusMsg_t`	A handle to a `NULL`-terminated string that a plug-in function returns to the Access Server to report on the result of the function.

Data Type/Name	Purpose
`void* ObAzPluginWritableData_t`	A handle to the head of the list of names for any of the read and write members of the opaque data structure ObAzPluginInfo.
`struct ObAzServerContext* ObAzServerContext_t`	A handle pointing to the head of the structure ObAzServerContext.

C Return Values

Many of the functions the Access Server and API use to communicate return a status value. These are all predefined, as described here.

ObAzplug-instatus_t

These are the possible values that plug-ins can return to show the result of the attempt to authorize.

```
typedef enum {
          ObAzplug-instatusContinue = 0,
          ObAzplug-instatusAccessAllowed = 1,
          ObAzplug-instatusAccessDenied = 2,
          ObAzplug-instatusAbort = 3
          }ObAzplug-instatus_t;
```

Value Name	Meaning
ObAzplug-instatusAbort	You indicate that a fatal error occurred within the plug-in. Processing is not passed to the following plug-in (if any). If this error is returned by an authorization plug-in, authorization fails and access is denied. If the error is returned by a custom action plug-in, an error message is logged, but authorization status is not affected. If returned during initialization, the Access Server logs an error message.
ObAzplug-instatusAccessAllowed	The plug-in authorizes access to the target by the requester. If the plug-in is an authorization plug-in, authorization processing stops and the Access Server moves on to success action processing. If the plug-in is a custom action plug-in, this response is ignored.
ObAzplug-instatusAccessDenied	The plug-in denies access to the target by the requester. If the plug-in is an authorization plug-in, authorization processing stops and the Access Server moves on to denied action processing. If the plug-in is a custom action plug-in, this response is ignored.
ObAzplug-instatusContinue	Authorization or custom action processing continues after the plug-in ends.

ObAzASStatus_t

When the plug-in calls SetDataFn to write data to the ObAzPluginInfo structure, the Access Server tries to do so, and returns one of these values.

```
typedef enum {

  ObAzASStatusSuccess = 0,
  ObAzASStatusWriteNotAllowed = 1
  }ObAzASStatus_t;
```

Value Name	Meaning
ObAzASStatusSuccess	The Access Server successfully performed the operation.
ObAzASStatusWriteNotAllowed	The Access Server did not perform the operation; specifically, the plug-in tried to change values it is not allowed modify.

C Structures

The Access Server groups related data items into named structures, allocates the memory for them, and holds the data that is in the structures. The API uses handles to read from and write data into these structures. Structures are opaque to the user, meaning that they can be used to transfer information to and from the Access Server, but the user cannot change the way the structure is organized or the format of the data that it contains. Following are the structures used in the Authorization Plug-in API.

ObAzServerContext

This structure carries information about the Access Server that the plug-in may need. It has two members.

```
const *ObAzserverContext{
   char *AccessServerInstallDir;
   char *AccessServerAzPluginAPIVersion;
};
```

The constant *ObAzServerContext_t is a handle pointing to the head of this structure.

Data held in the structure is read only.

The table below describes the structure members.

Data Type/Name	Purpose
`AccessServerInstallDir`	Path to the Installation directory for the NetPoint Access Server, for example NetPoint/AccessServer.
`AccessServerAzPluginAPIVersion`	The lowest Authorization Plug-in API version the Access Server currently supports.

ObAzPluginInfo

The Access Server fills this structure with data determined by the Authorization Scheme using the plug-in, in combination with the Authorization Rule that uses the scheme. The plug-in modifies data within the structure and may append new data to it as work progresses through the plug-in. When there are multiple authorization rules being carried out as part of a policy, the structure also provides a means to pass information from one plug-in to another within the rules.

```
struct ObAzPluginInfo{
   ObAzPluginData_tRequesterInfo;
   ObAzPluginData_tRequestContext;
   ObAzPluginData_tParams;
   ObAzPluginWritableData_tContext;
   ObAzPluginWritableData_tActionInfo;

};
```

The constant *ObAzPluginInfo_t is a handle pointing to this structure.

Data of type ObAzPluginWritableData_t can be both read and written. Data of type ObAzPluginData_t is read only.

Data is extracted from the structure and stored to it using the functions described under “ObAzPluginFns” on page 579.

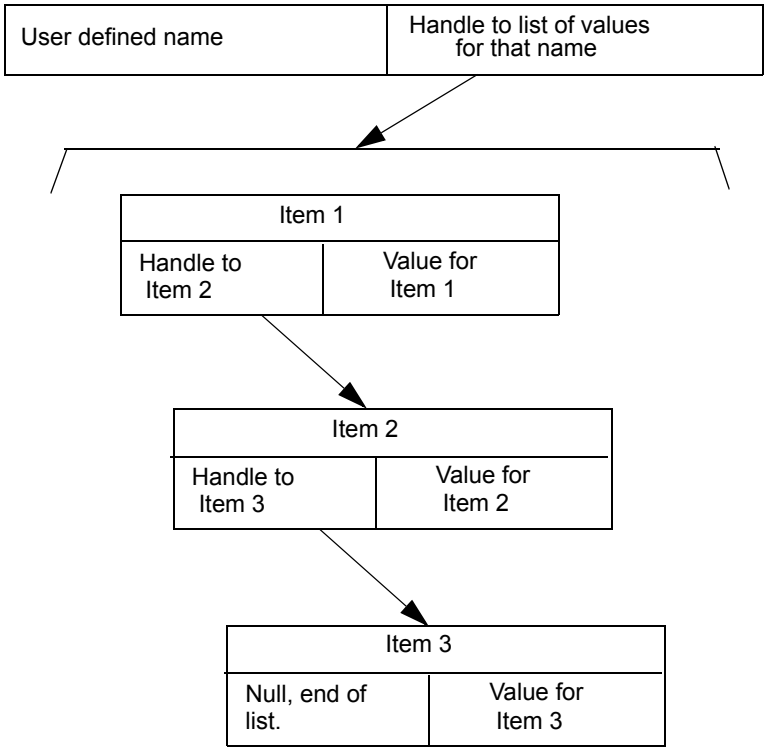
The table below describes the information provided by the members of this structure.

Data Type/Name	Purpose
`RequesterInfo`	Data describing the user or application trying to access a resource. The plug-in cannot change this data. The Access Server provides two predefined names within this list: RequesterDn and RequesterIP． Users can add other names as a User Parameter when the Authorization Scheme is configured. The parameter entered is the name of an attribute that can be found in the directory. The Access Server provides the name of the attribute and the value(s) of the entry for that attribute.
`RequestContext`	Request-specific information that is passed to the plug-in, such as a resource type. The plug-in cannot change this data. The Access Server provides three predefined names within this list: ResourceType, Resource, and Operation. Users cannot add others.
`Params`	Names and values for all required, optional, and additional parameters specified in the plug-in configuration. The plug-in cannot change this data. Names are created when an Authorization scheme is first created by the System or Master Access Administrator. Values can be provided when the scheme is first created, or added later by a Delegated Access Administrator when the scheme is used in an Authorization Rule.
`Context`	The plug-in uses this data to temporarily store or transfer information, for example to keep track of its own state when moving between logical modules, or to pass information to another plug-in. The plug-in may add new data or replace existing data. The Access Server maintains this data until the authorization request completes.
`ActionInfo`	The plug-in uses this data to return information, such as the authorization result, to the AccessGate. The plug-in may add new data or replace existing data. The Access Server maintains this data until the authorization request completes, when it is provided to the Client.

Understanding the organization of the ObAzPluginInfo structure is key to understanding how the Authorization Plug-in API works.

In the Authorization Plug-in API, all structure members are multivalued, meaning that the names held within each structure may each have more than one value. Each name has an associated handle that points to a list of one or more items. Each item contains a value and a handle to the next item in the list. A handle for the next item set to NULL indicates the end of the list.

The params member can be thought of according to the diagram below. You use GetDataFn to get the pointer to the list, for a specified parameter name within the params array.You then use GetFirstItemFn to get a handle to the information for the first item in the list. GetValueFn at this point returns the value for item 1, GetNextFn returns the handle to the information for item 2, and so on.



ObAzPluginFns

This structure provides handles to a block of functions provided by the Access Server, which the plug-in uses to manipulate data in the ObAzPluginInfo structure.

```
struct ObAzPluginFns
ObAzPluginFns_t{
ObASPluginGetFirstItem_tGetFirstItemFn;
ObASPluginGetValue_t  GetValueFn;
ObASPluginGetNext_t   GetNextFn;
ObAzPluginGetData_t   GetDataFn;
ObAzPluginsetData_t   SetDataFn;
};
```

The constant ObAzPluginFns_t is a handle pointing to this structure.

The table below describes the members of this structure. You may want to refer to “ObAzPluginInfo” on page 576 to understand how data is organized.

Note: This structure is a set of pointers to functions which are actually carried within the Access Server. You provide arguments for the function you want to use and a pointer to the function, and the Access Server returns the results.

Data Type/Name	Purpose
`GetFirstItemFn`	A function to get a handle for the first item in a list associated with a name.
`GetValueFn`	A function to read the value of an item, once the item’s handle has been defined using either GetFirstItemFn or GetNextFn.
`GetNextFn`	A function to get the handle for the next item in a list associated with a name. A returned handle value NULL means there are no more items in the list.
`GetDataFn`	A function to get a handle to a name in a specified member of the ObAzPluginInfo structure, given the text value for the name. A returned handle value NULL means that name is not present in that member of the structure.
`SetDataFn`	A function to store a value to a name in a specified member of the ObAzPluginInfo structure, given the text value for the name.

C API Functions

Functions used by the API to talk to the Access Server are of two types. They can be provided by the Access Server, in which case they are called by reference to it. Otherwise, they must be implemented in the plug-in, following the prototypes in authz_plugin_api.h.

C Functions Provided by the Access Server

These functions get and set data in the structures that have been passed. To use these functions you must call them as members of the structure of type ObAzPluginFns that you named in your code. For example, if you implemented ObAzPluginFn and set the variable name of type ObAzPluginFns to pFnBlock, then you call GetDataFn by reference to its place in the structure, as pFnBlock->GetDataFn.

GetDataFn

The plug-in uses this function to get a handle to the head of a list of values associated with a name in one of the members of the ObAzPluginInfo structure. The plug-in must then use the list manipulation functions GetFirstItemFn, GetValueFn, GetNextFn, GetValueFn, and so on, to extract information from the list. The function takes the form:

```
ObASPluginList_t GetDataFn(
   ObAzPluginData_t pmember,
   const char*      pName);
```

Input Parameters

Variable Name	Purpose
`pmember`	The member of the ObAzPluginInfo structure in which the name is expected to be found.
`pName`	The name for which values are to be retrieved.

Output Parameters

There are no output parameters for this function.

The function returns a handle to a list of values for the given name. If the handle value is NULL, the name is not present for the structure member.

SetDataFn

The plug-in uses this function to store a single value for a name in one of the members of the ObAzPluginInfo structure. The function takes the form:

```
ObAzASStatus_t SetDataFn(
   ObAzPluginData_t pMember,
   const char*      pName,
   const char*      pValue,
   const int        replace);
```

Input Parameters

Variable Name	Purpose
`pMember`	The name of a writable member of the ObAzPluginInfo structure.
`pName`	Name for the information whose value is to be set.
`pValue`	The value to be inserted.
`replace`	Specifies whether to replace or append to existing values for the name. A value of 0 indicates append, all other values are a request to replace the current first value for pName.

Output Parameters

There are no output parameters from this function.

The function returns one of the ObAzASStatus_t values.

Note: The replace option applies only to the first item in the list.

GetFirstItemFn

The plug-in uses this function to get a handle to the first item in a list of values, once the handle to the head of the list has been obtained using GetDataFn. You must then use the GetValueFn to extract the value or GetNextFn to get a handle to the next item in the list.

The function takes this form:

```
ObASPluginListItem_t GetFirstItemFn(
ObASPluginList_t plist);
```

Input Parameters

Variable Name	Purpose
`pList`	Handle to the head of a list of values, returned by GetDataFn.

Output Parameters

There are no output parameters for this function.

The function returns a handle to the first item in a list of values. If the handle value is NULL, there is no first item.

GetValueFn

The plug-in uses this function to get the value for an item, once a handle to the item has been obtained.

The function takes this form:

```
const char* GetValueFn(
     ObASPluginListItem_tPItem);
```

Input Parameters

Variable Name	Purpose
`pItem`	Handle to the current item in a list, returned by GetFirstItemFn or GetNextFn.

Output Parameters

There are no output parameters for this function.

The function returns the value of an item.

GetNextFn

The plug-in uses this function to get a handle to the next item in a list, given the handle to the current item.

The function takes this form:

```
ObASPluginListItem_t GetNextFn(
   ObASPluginListItem_tpItem);
```

Input Parameters

Variable Name	Purpose
`pItem`	Handle to the current item in a list, returned by GetFirstItemFn or GetNextFn.

Output Parameters

There are no output parameters for this function.

The function returns a handle to the next item in the list.

C Functions Implemented in the Plug-In

These functions describe the entry points that need to be in the .dll. Prototypes for these five functions are provided in authz_plugin_api.h. They must all be implemented in the plug-in.

The OBDLLEXPORT entry for each method is *required*. It provides a means for the Access Server to locate and call these methods from within the plug-in.

The Access Server calls the functions in this order:

- **GetVersion**—The first time the plug-in is loaded.
- **Init**—The first time the plug-in is loaded.
- **DeAllocStatusMessage**—Automatically, following any of the other functions which returns a status message.
- **Fn**—Each time the plug-in is used.
- **Terminate**—When the Access Server shuts down, or the plug-in is unloaded.

ObAzPluginGetVersion

The Access Server calls this function once when the plug-in is first loaded. The plug-in returns its version number, as defined in the authz_plugin_api.h file with which it was built. The Access Server uses this version to determine if it can support the plug-in. That is, it would catch a situation in which an older version of the Access Server was being asked to support a newer version of the API, or a newer version of the Access Server was being asked to support an obsolete version of the plug-in.

The function takes the form:

```
OBDLLEXPORT
   const char* ObAzPluginGetVersion(void)
```

Input parameters

There are no input parameters to this function.

Output parameters

There are no output parameters from this function.

The function returns the version of the authorization plug-in.

ObAzPluginInit

The Access Server calls this function after making the version check. You use ObAzPluginInit to initialize the workspace for the plug-in, which could include tasks such as connecting to a database and initializing global data for the plug-in. This function allocates memory in order to return the presult string, which must later be de-allocated using ObAzPluginDeallocStatusMsg.

The function takes the form:

```
OBDLLEXPORT
   ObAzplug-instatus_t ObAzPluginInit(
   ObAzServerContext_tpServerContext,
   ObAzplug-instatusMsg_tpResult)
```

Input Parameters

Variable	Purpose
`pServerContext`	The name assigned by the plug-in to the Context information structure provided by the Access Server.

Output Parameters

Variable	Purpose
`pResult`	Result message reported by the function.

The function must return one of two ObAzASStatus_t values, whose meaning is per the following table.

Status Name	Meaning
ObAzplug-instatusContinue	The workspace is successfully initialized. Processing of other plug-ins, of either type, if any, continues.
ObAzplug-instatusAbort	Initialization has failed. No additional plug-ins, of either type, are processed.

ObAzPluginTerminate

The Access Server calls this function when the Access Server terminates or the plug-in is unloaded. You use this function to clear the plug-in work area, for example to disconnect from a database or to free memory.

The function takes the form:

```
OBDLLEXPORT
   ObAzplug-instatus_t ObAzPluginTerminate(
   ObAzServerContext_tpServerContext,
   ObAzplug-instatusMsg_tpResult);
```

Input Parameters

Variable	Purpose
pServerContext	The name assigned by the plug-in to the Context information structure provided by the Access Server.

Output Parameters

Variable	Purpose
pResult	The result message reported by the function.

The function must return one of two ObAzASStatus_t values, whose meaning is per the following table.

Status Name	Meaning
ObAzplug-instatusContinue	The workspace is successfully cleared. Processing of other plug-ins, of either type, if any, continues.
ObAzplug-instatusAbort	The workspace could not be cleared. For example, a database connection could not be closed because the database was down. Processing of plug-ins ends.

ObAzPluginFn

The Access Server calls this function whenever a protected resource calls for authorization covered by a policy of which the plug-in is a part. You use this function to make the detailed decision or chain of decisions that determines whether access is denied or granted. The function defines either a custom authorization or a custom action process.

The function takes the form:

```
OBDLLEXPORT
     ObAzplug-instatus_t ObAzPluginFn(
     ObAzServerContext_tpServerContext,
     ObAzPluginFns_t     pFuncBlock,
     ObAzPluginInfo_t    pData)
```

Input Parameters

Variable	Purpose
`pServerContext`	The name you want to assign to the Context information structure provided by the Access Server.
`pFuncBlock`	Handle to the block of functions provided by the Access Server that the plug-in needs to manipulate data. You specify the name of this block.
`pData`	Handle to the ObAzPluginInfo structure in the Access Server. You specify the name of this structure.

Output Parameters

Variable	Purpose
`pData`	Handle to data modified by the plug-in.

The function returns one of the ObAzplug-instatus_t values, whose meaning is per the following table:

Status Name	Meaning
`ObAzplug-instatusContinue`	Regardless of the plug-in type, this signals the Access Server to move on to the next plug-in in the sequence. For an authorization plug-in, this means that the plug-in did not explicitly allow or deny access to the requester.
`ObAzplug-instatusAccessAllowed`	If this results from an authorization plug-in, the requester is allowed access to the target. The Access Server stops evaluating authorization plug-ins and moves on to success action plug-ins, if any. For a custom action plug-in this status is ignored.
`ObAzplug-instatusAccessDenied`	If this results from an authorization plug-in, the requester is denied access to the target. The Access Server stops evaluating authorization plug-ins and moves on to denied action plug-ins, if any. For a custom action plug-in this status is ignored.
`ObAzplug-instatusAbort`	Regardless of the plug-in type, processing does not continue after the function. If this results from an authorization plug-in, authorization fails.

ObAzPluginDeallocStatusMsg

The Access Server calls this function automatically when the plug-in terminates. You use it to delete the memory allocated by other plug-ins which returned a status message.

The function takes the form:

```
OBDLLEXPORT
     void ObAzPluginDeallocStatusMsg(
     ObAzplug-instatusMsg_t pStatusMsg);
```

Input Parameters

Variable	Purpose
`pStatusMsg`	Status Message to be deallocated.

Output Parameters

There are no output parameters from this function.

The function returns nothing

C Example

This example shows some basic uses of the plug-in functions. It is a modification of the authz_api.c sample function provided as part of the Access System Installation.

Listing 45 Authorization Plug-in Example.

```
OBDLLEXPORT const char* ObAzPluginGetVersion(void)
{
   FILE *file = fopen("d:\\AZtestfile.txt", "a+");
   fprintf (file, "\n%s %s\n", "getting version, it is",
      OB_AZ_PLUGIN_VERSION);
   fclose(file);
   return OB_AZ_PLUGIN_VERSION;
}

/*
* ------------------------------------------------
* Implementation of ObAnPluginInit
*
* The logged data appears only once, when the Plugin is first loaded.
*
*/

OBDLLEXPORT ObAzplug-instatus_t ObAzPluginInit(ObAzServerContext_t
pContext, ObAzplug-instatusMsg_t pResult)
{

// Values to be read in by this function are initialized.

   ObAzplug-instatus_t rtval;
   const char* pASPluginVersion = NULL;

   FILE *file = fopen("d:\\AZtestfile.txt", "a+");
   fprintf (file, "\n%s\n", "initializing");

   if(pContext != NULL) {
      pASPluginVersion = pContext->AccessServerAzPluginAPIVersion;
   }

   if((pASPluginVersion != NULL) &&
         (strcmp(pASPluginVersion, OB_AZ_PLUGIN_VERSION) == 0)) {
      rtval = ObAzplug-instatusContinue;
      *pResult = strdup("Success version check");
   } else {
/*
* return failure, because the version provided by the AS
* is not what was expected.
*/
      rtval = ObAzplug-instatusAbort;
   }

   fclose(file);
   return rtval;
}
```

Listing 46 Authorization plug-in example

```
/*
* ------------------------------------------------
* Implementation of ObAnPluginTerminate
*
* The logged data appears only when the Access Server terminates.
*
*/
OBDLLEXPORT ObAzplug-instatus_t ObAzPluginTerminate
      (ObAzServerContext_t pContext,
      ObAzplug-instatusMsg_t pResult)
{
   FILE *file = fopen("d:\\AZtestfile.txt", "a+");
   fprintf (file, "\n%s\n", "terminating gracefully");
   *pResult = strdup("Success, terminated");
   fclose(file);
   return ObAzplug-instatusContinue;
}
/*
* ------------------------------------------------
* Implementation of ObAnPluginDeallocStatusMsg
* The logged data appears following each other function
* that provides a presult.
*/
OBDLLEXPORT void ObAzPluginDeallocStatusMsg
      (ObAzplug-instatusMsg_t pResult)
{
   FILE *file = fopen("d:\\AZtestfile.txt", "a+");
   fprintf (file, "\n%s\n", "deallocating");

   if(pResult != NULL && *pResult != NULL) {
      free(*pResult);
      *pResult = NULL;
   }
   fclose(file);
}
/*
* ------------------------------------------------
* Implementation of ObAnPluginFn
*/
OBDLLEXPORT ObAzplug-instatus_t ObAzPluginFn
      (ObAzServerContext_t pContext,
      ObAzPluginFns_t pFnBlock,
      ObAzPluginInfo_t pInfo)
{
/*
* Default will be to continue without granting or denying
* authorization.
*/
   ObAzplug-instatus_t rtval = ObAzplug-instatusContinue;
```

Listing 46 (Continued) Authorization Plug-in Example.

```
* Pointers are defined.
*/
   ObASPluginList_t list;
   ObASPluginListItem_t item;
/*
* Data that might be read in is initialized.
*/
   const char* ou = NULL;
   const char* deny1 = NULL;
   const char* deny2 = NULL;
   const char* allow1 = NULL;
   const char* allow2 = NULL;
   const char* allow3 = NULL;
   const char* allow4 = NULL;
   int i = 0;
   FILE *file = fopen("d:\\AZtestfile.txt", "a+");
   fprintf (file, "\n%s\n", "doing real work");
   if((pFnBlock != NULL) && (pInfo != NULL)){
      fprintf (file, "%s\n", "first test okay, getting ou");
/*
* get user's "ou" from pInfo.
*/
      list = pFnBlock->GetDataFn(pInfo->RequesterInfo, "ou");
      item = pFnBlock->GetFirstItemFn(list);
      ou = pFnBlock->GetValueFn(item);
   }
/*
* show the ou value.
*/
   if(ou != NULL){
      fprintf (file, "%s\n", "ou was not null");
      fprintf (file,"%s %s \n", "ou is", ou);
   } else {
      fprintf (file, "%s\n", "ou was not found");
      rtval = ObAzplug-instatusAccessDenied;
      pFnBlock->SetDataFn
            (pInfo->ActionInfo, "access_status", "deny", 1);
      fclose(file);
      return rtval;
   )
/*
* now get two deny_organization values.
* This is risky coding, since it could be that "deny_organization"
* does not exist, or only has one value. In either case, the code
* will be generating NULL pointers, which could be misused elsewhere
*/
   list = pFnBlock->GetDataFn(pInfo->Params, "deny_organization");
```

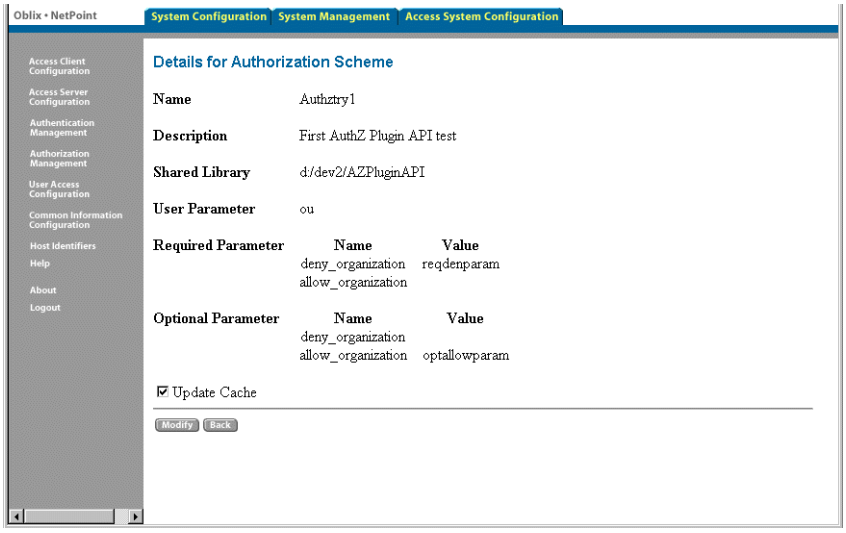
Listing 46 (Continued) Authorization Plug-in Example.

```
   if(list == NULL){
      fprintf (file, "%s\n", "missing deny org");
      rtval = ObAzplug-instatusAccessDenied;
      pFnBlock->SetDataFn
            (pInfo->ActionInfo, "access_status", "deny", 1);
      fclose(file);
      return rtval;
   }
   item = pFnBlock->GetFirstItemFn(list);
   deny1 = pFnBlock->GetValueFn(item);
   fprintf (file,"%s %s \n", "deny1 is", deny1);
   item = pFnBlock->GetNextFn(item);
   deny2 = pFnBlock->GetValueFn(item);
   fprintf (file,"%s %s \n", "deny2 is", deny2);
/*
* now get up to 4 allow_organization values.
*/
   list = pFnBlock->GetDataFn(pInfo->Params,"allow_organization");
   if(list == NULL){
      fprintf (file, "%s\n", "missing allow org");
      rtval = ObAzplug-instatusAccessDenied;
      pFnBlock->SetDataFn
            (pInfo->ActionInfo, "access_status", "deny", 1);
      fclose(file);
      return rtval;}
/*
* This is a better approach; it avoids generating null pointers.
*/
   for(i = 0, item = pFnBlock->GetFirstItemFn(list);
         item != NULL; i++, item = pFnBlock->GetNextFn(item)) {
      switch(i) {
         case 0:
            allow1 = pFnBlock->GetValueFn(item);
            fprintf (file,"%s %s \n", "allow1 is", allow1);
         break;
         case 1:
            allow2 = pFnBlock->GetValueFn(item);
            fprintf (file,"%s %s \n", "allow2 is", allow2);
         break;
         case 2:
            allow3 = pFnBlock->GetValueFn(item);
            fprintf (file,"%s %s \n", "allow3 is", allow3);
         break;
         case 3:
            allow4 = pFnBlock->GetValueFn(item);
            fprintf (file,"%s %s \n", "allow4 is", allow4);
         break;}}
```

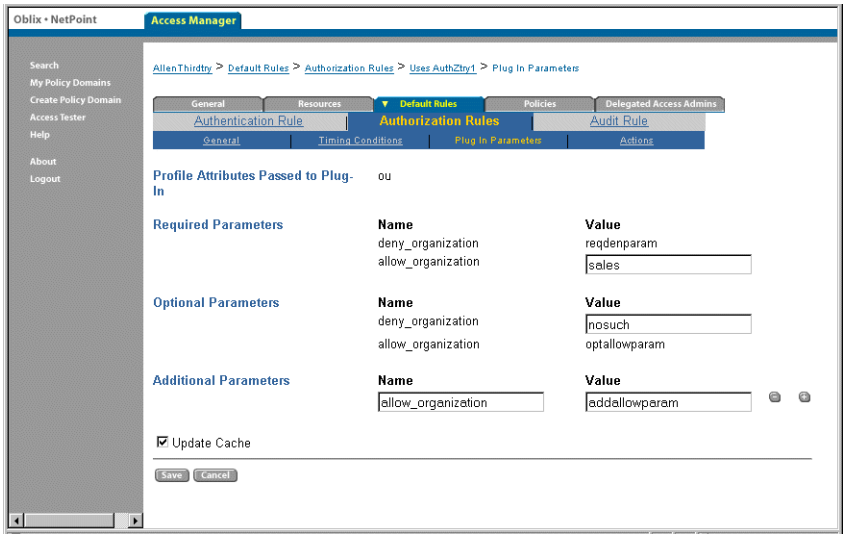
Listing 46 (Continued) Authorization Plug-in Example.

```
/*
* Look for a match between ou and one of the deny_organization
* values
*/
   if(((deny1 != NULL) && (strcasecmp(deny1, ou) == 0)) ||
             ((deny2 != NULL) && (strcasecmp(deny2, ou) == 0))) {
/*
* A match is found.
* Deny access to this user and set action information, using
* replace mode (last parameter = 1).
*/
      rtval = ObAzplug-instatusAccessDenied;
      pFnBlock->SetDataFn
            (pInfo->ActionInfo, "access_status", "deny", 1);
      fprintf (file,"%s \n", "access was denied");
      fclose(file);
      return rtval;
   } else {
/*
* Otherwise check for match between ou and one of the
* allow_organizations
*/
      if(((allow1 != NULL) && (strcasecmp(allow1, ou) == 0)) ||
         ((allow2 != NULL) && (strcasecmp(allow2, ou) == 0)) ||
         ((allow3 != NULL) && (strcasecmp(allow3, ou) == 0))) {
/*
* A match is found.
* Allow access to this user and set action information, using
* replace mode.
*/
            rtval = ObAzplug-instatusAccessAllowed;
            pFnBlock->SetDataFn
               (  pInfo->ActionInfo, "access_status", "allow", 1);
            fprintf (file,"%s \n", "access was allowed");
            fclose(file);
            return rtval;
      } else {
/*
* Otherwise no match was found in either category.
* Access is undefined by this plug-in.
*/
            rtval = ObAzplug-instatusContinue;
            pFnBlock->SetDataFn
               (pInfo->ActionInfo, "access_status",
                     "undefined", 1);
            fprintf (file,"%s \n", "access was undefined");
            fclose(file);
            return rtval;}}}
```

The following screen shows the initial setup for the Authorization Scheme, as set by the System Administrator. Note that space has been left for a required parameter, but no value has been entered.



Later, when a policy is defined to cover a resource that uses this Authorization scheme, the Delegated Access Administrator provides the missing required parameter value. In the example shown below, an additional value is also added.



Given this version of the authorization scheme, the corresponding trace information given by the sample code is:

```
getting version, it is 4.6

initializing

deallocating

doing real work
first test okay, getting ou
ou was not null
ou is Sales
deny1 is nosuch
deny2 is reqdenparam
allow1 is sales
allow2 is addallowparam
allow3 is optallowparam
access was allowed
```

Managed Code API Interfaces

Defines

The managed_plugin_interface.h file includes several defined values to aid in programming. One provides the value that is returned to the Access Server when ObAZPluginGetVersion is called:

```
#define OB-AZ_PLUGIN_VERSION "4.6"
```

Note: The value provided for the version may differ for later versions of NetPoint.

Interfaces

The Access Server and API use interfaces to allow manipulation of data structures that the Access Server maintains for use by the plug-in. These interfaces are named and described below.

Interface	Purpose
IObASPluginListItem	An interface that provides functions to access one of the items in a list of values.
IObAzPluginData	An interfaces that provides functions to access the list of read-only values.
IObAzPluginInfo	An interface that provides functions to access various data items that the plug-in can use.
IObAzPluginWritableData	An interface that provides functions to access and modify the list of values.
IObAzServerContext	An interface that provides functions to access server context information.

Return Values

Many of the functions that the Access Server and the API use to communicate return a status value. These are described in the following section.

Status

The following are the possible values that plug-ins can return to show the result of the authorization attempt.

```
IObAuthzPlugin::Status {
     ObAzpluginstatusContinue = 0,
        ObAzpluginstatusAccessAllowed = 1,
        ObAzpluginstatusAccessDenied = 2,
        ObAzpluginstatusAbort = 3
```

```
};
```

Value name	Description
ObAzpluginStatusAbort	Allows you to indicate that a fatal error occurred in the plug-in. Processing is not passed to the next plug-in (if there is any). If this error is returned by an authorization plug-in, authorization fails and access is denied. If the error is returned by a custom action plug-in, an error message is logged, but authorization status is not affected. If returned during initialization, the Access Server logs an error message.
ObAzpluginStatusAccessAllowed	The plug-in authorizes access to the target by the requester. If the plug-in is an authorization plug-in, authorization processing stops and the Access Server moves on to success action processing. If the plug-in is a custom action plug-in, this response is ignored.
ObAzpluginStatusAccessDenied	The plug-in denies access to the target by the requester. If the plug-in is an authroization plug-in, authorization processing stops and the Access Server moves on to denied action processing. If the plug-in is a custom action plug-in, this response is ignored.
ObAzpluginStatusContinue	Authoriztaion or custom action processing continues after the plug-in ends.

ASStatus

When the plug-in calls set_Data to write data to the ObAzPluginInfo structure, the Access Server tries to do this, and returns one of these values:

```
IObAuthzPlugin::ASStatus {
        ObAzASStatusSuccess = 0,
        ObAzASStatusWriteNotAllowed = 1
};
```

Value name	Description
ObAzASStatusSuccess	The Access Server successfully performed this operation.
ObAzASStatusWriteNotAllowed	The Access Server did not perform the operation. Specifically, the plug-in tried to change values that it is not allowed to modify.

Managed Code Interfaces

The Access Server groups related data items into structures and provides interfaces to access various members. The following are the interfaces used in the Authorization Plug-in API.

IObAzServerContext

This structure carries information about the Access Server that the plug-in may need. It has two members:

```
public _gc _interface IObAzServerContext
{
     _property String* get_AccessServerInstallDir();
     _property String* get_AccessServerAzPluginAPIVersion();
};
```

Data held in the structure is read-only.

The following table describes the functions provided by this interface.

Functions	Purpose
get_AccessServerInstallDir	Path to the installation directory for the NetPoint Access Server.
get_AccessServerAzPluginAPIVersion	The lowest Authorization Plug-in API version the Access Server currently supports.

IObAZPluginInfo

The Access Server fills this structure with data that is determined by the authorization scheme that uses the plug-in, along with the authorization rule that uses the scheme. The plug-in can modify data within the structure and may append new data to it as work progresses through the plug-in. When there are multiple authorization rules being carried out as part of a policy, the structure also provides a means to pass information from one plug-in to another within the rules.

```
public _gc _interface IObAzPluginInfo
{
     IObAzPluginData* GetRequesterInfo();
     IObAzPluginData* GetRequestContext ();
     IObAzPluginData* GetParams ();

     IObAzPluginWritableData* GetContext ();
     IObAzPluginWritableData* GetActionInfo ();
```

Data of type IObAzPluginWritableData can be both read and written. Data of type IObAzPluginData is read only.

The following table describes the information provided by the members of this structure:

Functions	Description
GetRequestInfo	Returns data that describes the user or application that is trying to access a resource. The plug-in cannot change this data. The Access Server provides two predefined names in this list: RequesterDN and RequesterIP. Users can add other names as a User Parameter when the authorization scheme is configured. The parameter entered is the name of an attribute that can be found in the directory. The Access Server provides the name of the attribute and the value(s) of the entry for that attribute.
GetRequestContext	Returns request-specific information that is passed to the plug-in, such as a resource type. The plug-in cannot change this data. The Access Server provides three predefined names in this list: ResourceType, Resource, and Operation. Users cannot add others.
GetParams	Returns names and values for all required, optional, and additional parameters specified in the plug-in configuration. The plug-in cannot change this data. Names are created when an authorization scheme is first created by the System or Master Access Administrator. Values can be provided when the scheme is first created, or added later by a Delegated Access Administrator when the scheme is used in an authorization rule.
GetContext	The plug-in uses this data to temporarily store or transfer information, for example to keep track of its own state when moving between logical modules, or to pass information to another plug-in. The plug-in may add new data or replace existing data. The Access Server maintains this data until the authorization request completes.
GetActionInfo	The plug-in uses this data to return information, such as the authorization result, to the AccessGate. The plug-in may add new data or replace existing data. The Access Server maintains this data until the authorization request completes, when it is provided to the client.

In the authorization plug-in API, all structure members are multi-valued, meaning that the names help within each structure may each have more than one value. Each name has an associated list of one or more items. Each item contains a value.

IObAzPluginData

This interface provides the functions to get the list of items.

```
public _gc _interface IObAzPluginData
{
     _property IEnumerator* get_Data(String* pName);
};
```

get_Data

The plug-in uses this function to get a list of values associated with a name in one of the members of the ObAzPluginInfo structure. The plug-in must then use the enumeration functions Current, MoveNext, and Reset to obtain items from the list.

Variable name	Purpose
pName	The name for which values are to be retrieved.

The function returns an object which implements the interface IEnumerator.

IObAzPluginWriteableData

This interface provides the functions to get and set the list of items.

```
public _gc _interface IObAzPluginWritableData
{
     IEnumerator* get_Data(String* pName);
     IObAuthzPlugin::ASStatus set_Data(String* key, String* val,
Oblix::ObListOper operation);
};
```

get_Data

The plug-in uses this function to get a list of values associated with a name in one of the members of the ObAzPluginInfo structure. The plug-in must then use the enumeration functions Current, MoveNext, and Reset to obtain items from the list.

Variable name	Purpose
pName	The name for which values are to be retrieved.

The function returns an object which implements the interface IEnumerator.

set_Data

The plug-in uses this function to store a single value for a name in one of the members of the ObAzPluginInfo structure. The function takes the following form:

Variable Name	Purpose
key	Name for the information whose value is to be set.
Val	The value to be inserted.
operation	Specifies whether to replace or append to existing values for the name. A value of ObListOper::ObAdd indicates append. All other values are a request to replace the current first value for key.

The function returns one of the ObAzASStatus_t values.

Note: The replace option applies only to the first item in the list.

IObAsPluginListItem

This interface provides the function to get the value of an item.

```
public _gc _interface IObASPluginListItem
     _property Sting* get_Value();
};
```

get_Value

The plug-in uses this function to get the value for an item once the item has been obtained. The function returns the value of an item.

Interfaces to be Implemented in the Plug-In

For authorization plug-ins the plug-in writer must define a class with the following functions:

```
namespace sample
{
     public _gc class ObAuthzPlugin
     {
     public:
          ObAuthzPlugin();
          String* ObAzPluginGetVersion();
          IObAuthzPlugin::Status ObAzPluginInit
(Oblix::IObAzServerContext* context, String* msg);
          IObAuthzPlugin::Status
ObAzPluginFn(Oblix::IObAzServerContext* context, Oblix::IObAzPluginInfo* info);
```

```
            IObAuthzPlugin::Status ObAzPluginTerminate
(Oblix::IObAzServerContext* context, String* msg);
            };
};
```

The class must be named ObAuthzPlugin, and may or may not be included in a namespace. All the functions need to have “public” access.

The Access Server calls the functions in this class in the following order:

- ObAzGetVersion—The first time the plug-in is loaded.
- ObAzPluginInit—The first time the plug-in is loaded.
- ObAzPluginFn—Each time the plug-in is used.
- ObAzPluginTerminate—When the Access Server shuts down.

ObAzPluginGetVersion

The Access Server calls this function once when the plug-in is first loaded. The plug-in returns its version number, as defined in the managed_plugin_interface.h file with which it was built. The Access Server uses this version to determine if it can support the plug-in. That is, it would catch a situation in which an older version of the Access Server was being asked to support a newer version of the API, or a newer version of the Access Server was being asked to support an obsolete version of the plug-in.

ObAzPluginInit

The Access Server calls this function after making the version check. You use ObAsPluginInit to initialize the workspace for the plug-in, which could include tasks such as connecting to a database and initializing global data for the plug-in.

Variable	Purpose
pServerContext	The name assigned by the plug-in to the Context information structure provided by the Access Server.

The function must return one of two ObAzASStatus values, describe in the following table:

Status name	Description
ObAzpluginStatusContinue	The workspace is successfully initialized.
ObAzpluginStatusAbort	Initialization has failed.

ObAzPluginTerminate

The Access Server calls this function when it terminates. You use this function to clear the plug-in work area, for example, to disconnect from a database.

Variable	Description
pServerContext	The name assigned by the plug-in to the context information structure provided by the Access Server.

The function must return one of two ObAzASStatus_t values, described in the following table:

Status name	Description
ObAzpluginStatusContinue	The workspace is successfully initialized.
ObAzpluginStatusAbort	Initialization has failed.

ObAzPluginFn

The Access Server calls this function when a protected resource calls for authorization covered by a policy of which the plug-in is a part. You use this function to make the detailed decision or chain of decisions that determines whether access is denied or granted. The function defines either a customer authorization or a custom action process.

Variable	Description
pServerContext	The name you want to assign to the context information structure provided by the Access Server.
pInfo	Handle to the ObAzPluginInfo structure in the Access Server. You specify the name of this structure.

The function returns one of the ObAzpluginstatus_t values, described in the following table:

Status name	Description
ObAzpluginStatusContinue	Regardless of the plug-in type, this signals the Access Server to move on to the next plug-in in the sequence. For an authorization plug-in, this means that the plug-in did not explicitly allow or deny access to the requester.
ObAzpluginStatusAccessAllowed	If this results from an authorization plug-in, the requester is allowed to access the target. The Access Server stops evaluating authorization plug-ins and moves on to denied action plug-ins, if there are any. For a custom action plug-in, this status is ignored.
ObAzpluginStatusAccessDenied	If this results from an authorization plug-in, the requester is denied access to the target. The Access Server stops evaluating authorization plug-ins and moves on to denied action plug-ins, if there are any. For a custom action plug-in this status is ignored.
ObAzpluginStatusAbort	Regardless of the plug-in type, processing does not continue after the function. If this results from an authorization plug-in, authorization fails.

Troubleshooting

For unit testing of plug-ins, writing the results to a file as the example here does is the best approach. The pResult text is captured only if authentication fails, and then only when the Access Server is running on Solaris. If you write to a file, be sure you have the correct permissions for writing into the directory holding the file.

Performance is a user responsibility and should be considered when designing a plug-in. The total time required to process one authorization request depends on the performance of all the plug-ins that are invoked while processing that request.

Plug-ins are trusted by the Access Server. No access check is performed when giving pre-configured information to the plug-in.

Coding errors at the system level in a plug-in, such as a memory or access violation, segmentation, or bus error fault, can crash the Access Server.

Plug-ins allow optional parameters, that would usually be filled in by a Delegated Administrator when schemes are created. Plug-ins should be able to gracefully handle the situation in which values for these parameters are not supplied.

If requests seem to fail without reason, check the path of the shared library to be sure it is correct.

A Oblix NetPoint Events

In the material on events and the custom actions you can write to handle them, beginning at “Identity Event Plug-in API” on page 171, the code examples introduce you to several NetPoint events. What was not described is how to find out what events are available, so that you can determine the best place to insert your custom code.

An *event* is defined as a significant point in the lifecycle of an interactive user request or workflow within NetPoint. Each NetPoint application defines its own events, and NetPoint automatically generates additional events to correspond with lifecycle steps in a Workflow.

Application Events

Currently, only the COREid System applications generate events. To determine which COREid System application generates which events, you need to examine the application registration files within the NetPoint installation. The registration files for the COREid System applications are found in the following locations:

Application	Registration File
(common to all)	<*$COREid_install_dir*>/apps/common/bin/oblixbasereg.xml
User Manager	<*$COREid_install_dir*>/apps/userservcenter/bin/userservcenterreg.xml
Group Manager	<*$COREid_install_dir*>/apps/groupservcenter/bin/groupservcenterreg.xml
Organization Manager	<*$COREid_install_dir*>/apps/objservcenter/bin/objservcenterreg.xml

See the chapter on PresentationXML in the *NetPoint 7.0 Customization Guide* for a discussion of how these files are used in a more general way.

In each registration file, you will find a set of XML elements named ObProgram. Each ObProgram element has a name attribute. The values of these name attributes are also the names of the NetPoint events generated by the COREid System applications.

Taking userservcenterreg.xml as an example, a snippet is

```
...
...
<ObProgram name="view">
   <ObStyleSheet name="usc_profile.xsl" />
   <ObButton name="initiateDeactivateUser" />
   <ObButton name="userreactivate" />
   <ObButton name="userModify" />
   <ObSchema name="usc_view.xsd" />
</ObProgram>
...
...
```

Notice the ObProgram element named view. View is also an event name, in this case the name of the event that corresponds to a page being displayed when you click on the My Identity (personal profile) tab in the User Manager.

Note: Another way to determine the name of an event, while interacting with NetPoint COREid System as an end user, is to examine the URL of the page currently being displayed. You will notice in the URL query string a pattern of the form: program=*xxxx*. When you see this, you can tell that the page was generated as the result of a NetPoint event called *xxxx*.

Workflow Events

In addition to application events, NetPoint generates events that you can associate with custom actions at predefined points in a workflow.

Workflow events are defined in the catalog in exactly the same way as application events. The difference is that their names are dynamically generated when the workflow steps are created. The name of a workflow event takes the following form:

<workflow>_<sequence>_<type>

where *workflow* is the name of the workflow, *sequence* is an integer representing the position of the step within the workflow, and *type* is either preaction or postaction.

B XML Background

This Appendix provides overviews of XML, XML schemas, and XSLT, for those who may need it in order to follow the discussion and examples for these topics in the main chapters of this Guide. Topics here include:

About XML

XML stands for Extensible Markup Language. It is a set of rules that define *tags* that break a document into parts and identify the parts of the document. These tags define a syntax that can then be used, in combination with an XSL stylesheet, to reconstruct the document.

The tags that are defined must follow the XML rules, but their content and arrangement can be anything the developer wants. A file of XML text, arranged to represent a certain document, is called an *XML application*. NetPoint's OutputXML is an XML application, designed to create HTML which will in turn present NetPoint pages to a browser.

NetPoint also uses XML as a structured way to provide some parameters that control its operation. This is a different use than for OutputXML, but since the applications are much shorter and the XML syntax rules are followed here as well, one of these files will serve as an example. For example, frontpageadminparams.xml has the following content:

```
<?xml version="1.0" ?>
      <ParamsCtlg xmlns="http://www.oblix.com"
            CtlgName="frontpageadminparams">
         <CompoundList ListName="">
            <SimpleList>
               <NameValPair ParamName="top_frame"
                  Value="_top" />
               <NameValPair ParamName="top_main_frame"
                  Value="main_frame" />
```

```
            <NameValPair ParamName="min_location_area"
               Value="400" />
         </SimpleList>
      </CompoundList>
   </ParamsCtlg>
```

This indented presentation, showing the tag levels, is an automatic feature of Microsoft's Internet Explorer. XML editors will also show the file in this way.

Some important parts of this file are the following:

```
<?xml version="1.0" ?>
```

This, the *XML declaration*, is the first line of any well-formed XML application. Internet Explorer and some editors will not show the file as formatted XML unless this line is present. The starting and ending ? make this an *XML processing instruction*. `version="1.0"` is an *attribute*. Attributes are name-value pairs separated by an equals sign, which provide additional information for the instruction. Currently there is only one version of XML.

```
<ParamsCtlg xmlns="http://www.oblix.com"
            CtlgName="frontpageadminparams">
```

<ParamsCtlg> is a tag, which starts the definition of the first element in the XML application. The definition ends with the matching closing tag, which has the same form except it uses a / before the tag name:

```
</ParamsCtlg>
```

Everything between the starting and ending tags defines the element `ParamsCtlg`. Nested within it is the element CompoundList, which has elements nested within it, and so on. An important attribute is xmlns, which stands for *XML namespace.* This specifies an owner and possible reference source for this XML application. We identify ourselves as creators of this application.

```
  <NameValPair ParamName="top_frame" Value="_top" />
```

The technically precise way to write this element would have been

```
    <NameValPair>
         ParamName="top_frame" Value="_top"
    </NameValPair>
```

However, when the definition is a short one like this, the XML rules allow use of an abbreviated closing tag. /> indicates the closing tag for the immediately preceding start tag.

The attributes ParamName="top_frame" and Value="_top" provide the useful content of the file, which is the name of a variable used by NetPoint and its value.

An important concept, essential to the application of stylesheets, is a *node*. A node is a level within the XML application, described by stringing together the elements that locate it uniquely within the nested elements. For example, `ParamsCtlg` is the *root* node for the application. The root node is the element name immediately following the XML processing instruction(s); all other elements are nested within it. Other examples of nodes are ParamsCtlg/CompoundList and ParamsCtlg/CompoundList/SimpleList.

XML Schema

An XML Schema shows and describes the content of an XML application. The following list interprets some of the elements that appear within a NetPoint Schema definition file, based on the first few characters of each element. This is *not* intended to be an explanation of the full XML Schema syntax; see the referenced site for that.

xsd:attribute—Appears within the body of an element being defined, and defines an attribute that belongs to it. Parts of the definition usually present are:

- **name=”*xxxx*”**—The name of the attribute
- **type=”*yyyy*”**—The data type for the attribute; see the list below
- **use=”required”**—This is present only if the attribute must be present in the output.
- **value=”*zzzz*”**—This is present only if the attribute takes a fixed value.

xsd:choice—Precedes a list of other elements, indicating that one and only one of those elements is allowed. The choice itself can be made from zero to many times, as controlled by the values of minOccurs and maxOccurs. The value of minOccurs is the fewest number of times this element can appear in the list. If the value is zero, the element is optional in the list. The value of maxOccurs is the greatest number of times the element can appear in the list. A value of Unbounded means there is no limit.

xsd:complexType—Most often used in the body of an element that is being defined, and means that the element will contain other elements.

xsd:element name=”*xxxx*”—Declares and within its body goes on to fully define a category of information describing the element *xxxx*. Most instances of this in the schema files go on to provide a body for the element and build it up from subelements. A few, for example ObTextMesage in the displaytype.xsd file, have no body, in which case they use type to immediately specify the data type of the element.

xsd:element ref="*xxxx*"—Most often used to provide the name of a subelement for inclusion in a list that is part of the body defining an element. The referenced element will have been defined elsewhere. The element may also include the attributes minOccurs and maxOccurs.

xsd:enumeration—Provides a list of possible values.

xsd:include schemalocation="*xxxx*"—An element that specifies a file which contains additional XML schema information, to be treated just as if it were provided inline in the current file.

xsd:restriction base="*xxxx*"—Defines the pattern for values that are used for a data type being defined; see xsd:simpletype. NetPoint uses the restriction base NMTOKEN, which means the value must be a legal XML string and contain no white spaces.

xsd:sequence—Precedes a list of subelements within another element, and indicates that, if they are present, they will appear in the order listed.

xsd:simpletype—This begins the definition of a data type, usually followed by an xsd:restriction definition.

Some possible data types are:

xsd:boolean—Acceptable values are true/false, or 1/0.

xsd:date—Acceptable values are dates in the form YYYY-MM-DD (many other date types are possible).

xsd:decimal—Acceptable values are decimal numbers (other number types are possible)

xsd:string—Acceptable values are a string of characters

xsd:time—Acceptable values are a time of day in the form hh:mm:ss.sss.

xsd:uri-reference—Acceptable values are URLs.

All of the Oblix XML schemas are defined within a root element called oblix. The table below shows the schema for the usc_profile.xsd definition of oblix, beginning with its initial definition in component_profile.xsd. The table shows the schema only to the first two node levels below oblix; the full schema goes much deeper. If you look at just the pure OutputXML provided by NetPoint for the view (My Identity) program, this information, in this order, is what you see.

Level 1	Level 2
ObProfile (defined in component_profile.xsd)	ObPanel
	ObHeaderPanel
	ObRequestInfo
	ObScripts
	ObForm
	ObDisplay
	ObTextMessage
	ObButton
	ObStatus
ObNavBar (defined in navbar.xsd)	ObRequestInfo
	ObScripts
	ObMisc
	ObApps
	ObApplication
	ObFunctionsButtons
	ObStatus
ObSearchForm (defined in searchform.xsd)	ObHelpContext
	ObRequestInfo
	ObScripts
	ObForm
	ObDisplay
	ObButton
	ObAdvancedSearch
	ObSearchRow
	ObStatus

Level 1	Level 2
ObApplicationFunc (defined in navbar.xsd)	ObFunctions
	ObRequestInfo
	ObStatus
ObStatus (defined in component_basic.xsd)	This is a string of type xsd:string; it contains no other elements.

XSL and XSLT

XSL stands for Extensible Style Language. Files written in this language are used along with XSLT to create documents. The XSL file itself is a well-formed XML document. The language relies heavily upon the use of *templates*, which are sets of instructions to the XSL transformer, telling it what to produce as output for a particular node within the XML.

XSLT stands for XSL Transformation. This is a process that combines an XML application with an XSL stylesheet to create a document.

General Syntax

The following list interprets some of the elements that appear within a NetPoint stylesheet file, based on the first few characters of each element. This is *not* intended to be an explanation of the full XSLT syntax; see the referenced site for that.

Note: In NetPoint's XSL files, lines starting with <xsl: are instructions to the XSL transformer. All others are HTML text to be written verbatim into the HTML output.

---

xsl:apply-templates select="*xxxx*"—Once the transformer is positioned to a node within the XML, using xsl:template-match, this element identifies which subnodes or sub-subnodes are to be processed. Point at sub-subnodes within the selected node by providing their nested structure, for example xsl:apply-templates="*xxxx*/*yyyy*", where *yyyy* is a node nested within *xxxx*. If the select option is omitted, templates for all the subnodes under the matched node are processed.

The transformer decides which templates to use by identifying each subnode by name, and then searching the entire stylesheet for the best xsl:template match for that name. The match will generally be on the last node in the nested list, for example *yyyy* in the above example. The instructions for that matched node are applied immediately.

xsl:attribute name=”*string*”—Inserts the text specified by string into the output.

xsl:call-template name=”*xxxx*”—Immediately performs the transformation required by the template *xxxx*. The template to be called will have been specified using xsl:template name=”*xxxx*”.

xsl:choose—Precedes a list of possible transformations, each of which is indicated by the use of the xsl:when element. It may be that none of the xsl:when elements applies; the xsl:otherwise element covers this possibility. If more than one of the xsl:when elements is true, only the first true xsl:when element is applied.

xsl:for-each select=”*xxxx*”—Applies the content of this element to all occurrences of *xxxx*.

xsl:if test=”expression”—Allows a choice to be made. If expression evaluates to a Boolean true, the content of the xsl:if element is performed. If not, it’s not performed. Expression syntax is described below.

xsl:include href =”*xxxx*”—An element that specifies a file which contains additional XSL stylesheet information, to be treated just as if it were provided inline in the current file.

xsl:number value=”*expression*”—Used to insert a formatted integer into the output. In Oblix stylesheets, expression often uses the position() function, which indicates the position of a node in a list, starting with 1.

xsl:otherwise—The last element in the list of elements under an xsl:choose, following the xsl:whens, which is to be applied if none of the xsl:whens is true.

xsl:template match=”*xxxx*”—Point the transformer to the node named xxxx in the XML data. Point at subnodes by providing their nested structure, for example xsl:template-match=”*xxxx/yyyy*”, where *yyyy* is a node nested within xxxx. This must be followed by one or more uses of xsl:apply-templates, otherwise no transformation of the XML data will be done.

xsl:template name=”*xxxx*”—Create a named template, to be applied when xsl:call-template=”*xxxx*” is used.

xsl:value-of select=”*expression*”—Inserts the value specified by *expression* into the output.

xsl:when test=”*expression*”—Allows a choice to be made. If *expression* evaluates to a Boolean true, the content of the xsl:when element is performed. If not, it’s not performed. Usually, multiple xsl:when elements are nested under an xsl:choose element.

Expression Syntax

Again, this is only a subset of a much longer list, provided to allow you to interpret NetPoint XSL files. Expressions can be of several kinds:

- Node Sets

  A node set describes a set of nested elements, in the form *xxxx*/*yyyy*/*zzzz*, meaning the element *zzzz* is nested within the element *yyyy* which is then nested within the element *xxxx*. When a node set is used as the expression for a test, the test is true if the nested set exists in the XML, false if it does not.

  Further, this may be used in the form *xxxx*/*yyyy*/*zzzz*[@attribute = *a value*]. This means to look at the value of the attribute belonging to element *zzzz*. The expression is true if the attribute has the specified value and false otherwise.

- String Content

  One form of this is

  ```
  <xsl:value-of select="@attribute" />
  ```

  which means return the value of the attribute.

  Another is

  ```
  <xsl:if test="@attribute">
  ```

  which is true if the attribute is valid for the element and has a non-NULL value.

- Numeric Content

  In this case, the expression reduces to a number. An example is

  ```
  <xsl:number value="position()-1">
  ```

  which gives a number one less than the position of the current element in a list of elements.

Client-Side Transformation

Client-side processing of stylesheets is supported only with Microsoft Internet Explorer (IE) 5.0 and later. Earlier versions of IE require installation of a patch..

To set up client side processing

1. Install the latest msxml patch.

   This must be msxml3.0 (or above), which can be obtained from:

```
http://download.microsoft.com/downloads/xml/Install/3.0/WIN98Me/EN-US/
msxml3.exe
```

2. Install the registration tool for msxml.

   This can be obtained from:

```
http://msdn.microsoft.com/msdn-files/027/001/469/xmlinst.exe
```

3. Enter the following command sequence:

   ```
   xmlinst -u

   regsvr32 -u msxml.dll

   regsvr32 msxml3.dll

   xmlinst
   ```

4. Change the controlling parameter.

   In the *$COREid_install_dir*/apps/common/bin/globalparams.xml parameter file, change the value for OutputFormat from default to xml.

5. Restart the COREid Server.

6. Verify the change.

   To verify that this change indeed took place, enter the COREid System using an Internet Explorer 5 browser.

   If you do a view source, you will see XML instead of HTML.

NetPoint XSL Transformation Limits

NetPoint has a built in XSL Transformation processor. This processor implements most, but not all, of the XSLT standard. The following is some information applying to the NetPoint version.

- The processor does not automatically insert the declaration line:

  ```
  <?xml version="1.0" ?>
  ```

  in XML files that it generates. If this is needed because you want to see an indented XML presentation, you must include it in the stylesheet.

- The processor does not support UTF characters in a sort. An attempt to do this will generate an error report.

- The processor has a stack limit depth of 5298; recursive templates can go no deeper than this.

- The processor assumes that its output is intended for use by a browser and therefore formats output with an HTML formatter.

- The processor is intended primarily for use in a production environment, where performance is important. For this reason, it does only minimal checking of stylesheet syntax. Very bad syntax can crash the processor. For this reason, only known stylesheets with validated content should be used in the production environment. Some validation tools are listed in the *NetPoint 7.0 Customizaton Guide*.

- Embedded stylesheets in the XML are not supported.
- Full support, or in some cases, any support, of the following commands is not provided. If you need to use these commands, double-check your results before putting the stylesheet into production.
  - XSL:format-number
  - XSL:output
  - XSL:document
  - XSL:namespace
  - XSL:comment
  - XSL:format
  - XSL:processing instruction
  - XSL:sort—case order
  - XSL: id

Resources

Full descriptions and specifications for this information are available at:

www.ww3.org, under XML, XML Schema, and XSL

Find documentation for XML at:

http://www.w3.org/XML/

Find documentation for XML Schema at:

http://www.w3.org/XML/Schema

A tutorial on the XML schema syntax is available at:

http://www.w3.org./TR/xmlschema-0

Find documentation for XSL and XSLT at:

http://www.w3.org/Style/XSL/
http://www.w3.org/TR/xslt

C Access Management API Definitions

This appendix contains definitions for the Java, C, and managed code application programming interfaces referred to as the Access Management API:

- “Class ObAccessManager” on page 622
- “Access Policy Objects” on page 626
- “Access System Configuration Objects” on page 658
- “Class ObAMException” on page 663

The interfaces are defined here for easy reference. For complete header file content, refer to:

- Javadocs for Java definitions.
- The obaccessmgr_api.h file for C definitions.
- The obaccessmgr_api_mgd.h and the obaccess_api_common_mgd2.h file for managed code definitions.
- For detailed information about the Access Management API, see “Access Management API” on page 403.

Class ObAccessManager

For details see:

- "Java" on page 622
- "C" on page 623
- "Managed Code" on page 624

Java

```
public class ObAccessManager {
// INITIALIZATION METHODS
   public ObAccessManager();
   public void setAdmin(
         String userid,
         String password) throws ObAMException;
   public void setCacheUpdates(bool update);
// GET METHODS
   public Object[] getObjects(
   int responselength,
   String matchName,
   int matchCriterium) throws ObAMException;
// RESPONSE LENGTHS
   public static final int MIN = 0;
   public static final int MID = 1;
   public static final int MAX = 2;
// MATCH CRITERIA
   public static final int EQUALS      = 0;
   public static final int CONTAINS    = 1;
   public static final int CONTAINS_IN_ORDER = 2;
   public static final int BEGINS_WITH = 3;
   public static final int ENDS_WITH   = 4;
   public ObAMPolicyDomain[] getPolicyDomains(
         int responselength,
         String matchName,
         int matchCriteria) throws ObAMException;
   public ObAMAuthenticationScheme[]
   getAuthenticationSchemes(
         int responselength, String matchName,
         int matchCriteria) throws ObAMException;
   public ObAMAuthorizationScheme[]
   getAuthorizationSchemes(
         int responselength, String matchName,
         int matchCriteria) throws ObAMException;
   public ObAMResourceType[] getResourceTypes(
         int responselength,
         String matchName,
         int matchCriteria) throws ObAMException;
   public ObAMHostIdentifier[] getHostIdentifiers(
         int responselength,
```

```
        String matchName,
        int matchCriteria) throws ObAMException;
// SET METHOD AND SET ACTIONS
   public static final int CREATE = 0;
   public static final int MODIFY = 1;
   public static final int REMOVE = 2;
   public void setPolicyDomain(
        ObAMPolicyDomain value,
        int setAction) throws ObAMException;
//ACCESS TEST METHOD
   public ObAMAccessTestResults testAccess(
        ObAMAccessTest test);
}
```

C

```
/* INITIALIZATION METHODS */
typedef const void * ObAccessManager_t;
   ObAccessManager_t ObAccessManager_new();
   void ObAccessManager_delete(ObAccessManager_t *pAm);
   void ObAccessManager_setAdmin_password(
        ObAccessManager_t am,
        const char *userid,
        const char *password)
   void ObAccessManager_setCacheUpdates(int update);
/* GET METHODS */
   enum ObAccessManager_ResponseLength {
        ObAccessManager_MIN,
        ObAccessManager_MID,
        ObAccessManager_MAX};
   enum ObAccessManager_MatchCriteria {
        ObAccessManager_EQUALS,
        ObAccessManager_CONTAINS,
        ObAccessManager_CONTAINS_IN_ORDER,
        ObAccessManager_BEGINS_WITH,
        ObAccessManager_ENDS_WITH};
ObAMArrayOfObjects_t ObAccessManager_getObjects(
        ObAccessManager_t am,
        ObAccessManager_ResponseLength responseLength,
        const char *matchName,
        ObAccessManager_MatchCriteria matchCriterium);
ObAMArrayOfPolicyDomains_t
   ObAccessManager_getPolicyDomains(
        ObAccessManager_t am,
        ObAccessManager_ResponseLength responseLength,
        const char *matchName,
        ObAccessManager_MatchCriteria matchCriterium);
ObAMArrayOfAuthenticationSchemes_t
   ObAccessManager_getAuthenticationSchemes(
        ObAccessManager_t am,
        ObAccessManager_ResponseLength responseLength,
        const char *matchName,
```

```
            ObAccessManager_MatchCriteria matchCriterium);
   ObAMArrayOfAuthorizationSchemes_t
      ObAccessManager_getAuthorizationSchemes(
            ObAccessManager_t am,
            ObAccessManager_ResponseLength responseLength,
            const char *matchName,
            ObAccessManager_MatchCriteria matchCriterium);
   ObAMArrayOfResourceTypes_t
      ObAccessManager_getResourceTypes(
            ObAccessManager_t am,
            ObAccessManager_ResponseLength responseLength,
            const char *matchName,
            ObAccessManager_MatchCriteria matchCriterium);
   ObAMArrayOfHostIdentifiers_t
      ObAccessManager_getHostIdentifiers(
            ObAccessManager_t am,
            ObAccessManager_ResponseLength responseLength,
            const char *matchName,
            ObAccessManager_MatchCriteria matchCriterium);
   ObAMMasterAuditRule_t
      ObAccessManager_getMasterAuditRule(
            ObAccessManager_t am);
   /* SET METHOD */
   enum ObAccessManager_SetAction {
            ObAccessManager_CREATE,
            ObAccessManager_MODIFY,
            ObAccessManager_REMOVE};
      void ObAccessManager_setPolicyDomain(
            ObAccessManager_t am,
            ObAMPolicyDomain_t value,
            ObAccessManager_SetAction setAction);

   /* ACCESS TEST METHOD */
   ObAMAccessTestResults_t ObAccessManager_testAccess(
            ObAMAccessTest_t test);
```

Managed Code

```
public __gc class ObAccessManager_ResponseLengthMgd {
   public:
      ObAccessManager_ResponseLengthMgd();
   // GET AND SET VALUES
      __property bool get_isMin();
      __property bool get_isMid();
      __property bool get_isMax();
      __property ObAccessManager_ResponseLength get_Value();
      __property void set_Value(ObAccessManager_ResponseLength value);
      void setMin();
      void setMid();
      void setMax();
};
```

```
public _gc class ObAccessManager_MatchCriterMgd {
   public:
      ObAccessManager_MatchCriteriaMgd():
   //GET AND SET VALUES
   __property bool get_isEquals();
   __property bool get_isContains();
   __property bool get_isContainsInOrder();
   __property bool get_isBeginsWith():
   __property bool get_isEndsWith();
   __property ObAccessManager_MatchCriteria get_Value();
   __property void set_Value(ObAccessManager_MatchCriteria value);
   void setEquals();
   void setContains();
   void setContainsInOrder();
   void setBeginsWith();
   void setEndsWith();
};


public _gc class ObAccessManager_SetActionMgd {
   public:
      ObAccessManager_SetActionMgd();
   // GET AND SET VALUES
      __property bool get_isCreate():
      __property bool get_isModify();
      __property bool get_isRemove();
      __property ObAccessManager_SetAction get_Value();
      __property void set_Value(ObAccessManager_SetAction value);
      void setCreate();
      void setModify();
      void setRemove();
};



   // INITIALIZATION
public _gc class ObAccessManagerMgd : public System::IDisposable {
   public:
      ObAccessManagerMgd();
      ~ObAccessManagerMgd();
      void Dispose();
      void Dispose(bool disposing);
   // GETTERS AND SETTERS
      void setAdmin(System::String *userid, System::String *password);
      __property void set_CacheUpdates(bool update);
   // RETURNS AN ARRAY OF ObAMPolicyDomainMgd OBJECTS
   ArrayList
   *ObAccessManagerMgd::getPolicyDomains(ObAccessManager_ResponseLengthMgd
           *responseLength, System::String *matchName,
           ObAccessManager_MatchCriteriaMgd *matchCriterium);
   // RETURNS AN ARRAY OF ObAMAuthenticationSchemeMgd OBJECTS
      ArrayList *getAuthenticationSchemes(ObAccessManager_ResponseLengthMgd
           *responseLength, System::String *matchName,
```

```
        ObAccessManager_MatchCriteriaMgd *matchCriterium);
// RETURNS AN ARRAY OF ObAMResourceTypeMgd OBJECTS
   ArrayList *getResourceTypes(ObAccessManager_ResponseLengthMgd
        *responseLength, System::String *matchName,
        ObAccessManager_MatchCriteriaMgd *matchCriterium);
// RETURNS AN ARRAY OF ObAMHostIdentifierMgd OBJECTS
   ArrayList *getHostIdentifiers (ObAccessManager_ResponseLengthMgd
        *responseLength, System::String *matchName,
        ObAccessManager_MatchCriteriaMgd *matchCriterium);
   __property bool get_isContains();
   ObAMMasterAuditRuleMgd *get_MasterAuditRule();
   void setPolicyDomain(ObAMPolicyDomainMgd *value,
        ObAccessmanager_SetActionMgd *setAction);
   ObAMAccessTestResultsMgd *getTestAccess(ObAMAccessTestMgd *test);
};
```

Access Policy Objects

Refer to the discussions below for details:

- "Java" on page 626
- "C" on page 635
- "Managed Code" on page 648

Java

Class ObAMResource

```
public class ObAMResource {
   public ObAMResource();
   public String getResourceType();
   public String getHostID();
   public String getURLPrefix();
   public String getDescription();
   public void setResourceType(String value);
   public void setHostID(String value);
   public void setURLPrefix(String value);
   public void setDescription(String value);
   public void setIDFrom(ObAMResource other);
}
```

Class ObAMAccessConditions

```
public class ObAMAccessConditions {
   public ObAMAccessConditions();
   public int getNumberOfPersons();
   public int getNumberOfGroups();
```

```
   public int getNumberOfRoles();
   public int getNumberOfRules();
   public int getNumberOfIPaddresses();
   public ObAMIdentity_t getPerson(int index);
   public ObAMIdentity_t getGroup(int index);
   public String getRole(int index);
   public String getRule(int index);
   public String getIPaddress(int index);
   public void addRole(String value);
   public void addPerson(ObPerson value);
   public void addGroup(ObGroup value);
   public void addRule(String value);
   public void addIPAddress(String value);
   public void removeRole(String value);
   public void removePerson(ObPerson value);
   public void removeGroup(ObGroup value);
   public void removeRule(String value);
   public void removeIPAddress(String value);
   public void setIDFrom(ObAMAccessConditions other);
}
```

Class ObAMDate

```
public class ObAMDate {
// DAYS OF THE WEEK
   public static final int SUNDAY    = 1;
   public static final int MONDAY    = 2;
   public static final int TUESDAY   = 3;
   public static final int WEDNESDAY = 4;
   public static final int THURSDAY  = 5;
   public static final int FRIDAY    = 6;
   public static final int SATURDAY  = 7;
// MONTHS
   public static final int JANUARY   = 1;
   public static final int FEBRUARY  = 2;
   public static final int MARCH     = 3;
   public static final int APRIL     = 4;
   public static final int MAY       = 5;
   public static final int JUNE      = 6;
   public static final int JULY      = 7;
   public static final int AUGUST    = 8;
   public static final int SEPTEMBER = 9;
   public static final int OCTOBER   = 10;
   public static final int NOVEMBER  = 11;
   public static final int DECEMBER  = 12;
   public ObAMDate();
   public int getYear();
   public int getMonth();
   public int getDay();
   public void set(int year, int month, int day)
         throws ObAMException;
}
```

Class ObAMTime

```
public class ObAMTime {
   public ObAMTime();
   public int getHours();
   public int getMinutes();
   public int getSeconds();
   public void set(int hours, int minutes,int seconds)
         throws ObAMException;
}
```

Class ObAMTimingConditions

```
class ObAMTimingConditions {
   // VALUE FOR RelativeTo
   public static final int UNDEFINED  = 0;
   public static final int GMT        = 1;
   public static final int LOCAL_TIME = 2;
   ObAMTimingConditions();
   public int getRelativeTo();
   public ObAMDate getStartDate();
   public ObAMTime getStartTime();
   public ObAMDate getEndDate();
   public ObAMTime getEndTime();
   public int getNumberOfMonths();
   public int getNumberOfDaysOfMonth();
   public int getNumberOfDaysOfWeek();
   public int getMonth(int index);
   public int getDayOfMonth(int index);
   public int getDayOfWeek(int index);
   public void setRelativeTo(
         int value) throws ObAMException;
   public void setStartDate(ObAMDate value);
   public void setStartTime(ObAMTime value);
   public void setEndDate(ObAMDate value);
   public void setEndTime(ObAMTime value);
   public void addMonth(
         int value) throws ObAMException;
   public void addDayOfMonth(
         int value) throws ObAMException;
   public void addDayOfWeek(
         int value) throws ObAMException;
   public void removeMonth(
         int value) throws ObAMException;
   public void removeDayOfMonth(
           int value) throws ObAMException;
   public void removeDayOfWeek(
           int value) throws ObAMException;
   public void setIDFrom(ObAMTimingConditions other);
}
```

Class ObAMIdentity

```
public class ObAMIdentity {
   public ObAMIdentity();
   public String getUID();
   public String getName();
   public String getLoginID();
   public void setUID(String value);
   public void setName(String value);
   public void setLoginID(String value);
}
```

Class ObAMObjectWithActions

```
public class ObAMObjectWithActions {
// ACTION TYPE ENUMS
   public static final int SUCCESS = 0 ;
   public static final int FAILURE = 1 ;
   public static final int INCONCLUSIVE = 2 ;
   public int getNumberOfActions(int actionType)
         throws ObAMException;
   public ObAMAction getActionOfType(int actionType, int index)
         throws ObAMException;
   public void addActionOfType(int actionType, ObAMAction value)
         throws ObAMException;
   public void removeActionOfType(int actionType,
         ObAMAction value) throws ObAMException;
   public String getName();
   public void setName(String value);
   public void setIDFrom(ObAMObjectWithActions other);
}
```

Class ObAMAction

```
public class ObAMAction {
   public static final int UNDEFINED   = 0;
   public static final int FIXEDVALUE  = 1;
   public static final int ATTRIBUTE   = 2;
   public ObAMAction();
   public String getType();
   public String getName();
   public String getValue();
   public int getValueType();
   public void setType(Stromg value);
   public void setName(String value);
   public void setValue(String value);
   public void setValueType(
         int value) throws ObAMException;
   public void setIDFrom(ObAMAction other);
}
```

Class ObAMAuthenticationRule

```
public class ObAMAuthenticationRule {
   public ObAMAuthenticationRule();
   public String getName();
   public String getDescription();
   public String getScheme();
   public void setName(String value);
   public void setDescription(String value);
   public void setScheme(String value);
   public void setIDFrom(
         ObAMAuthenticationRule other);
   public int getNumberOfActions(int actionType)
         throws ObAMObjection;
   public ObAMAction getActionOfType(int actionType,
         int index)throws ObAMException;
   public void addActionOfType(int actionTtype,
         ObAMAction value)throws ObAMException;
   public void removeActionOfType(int actionType,
         ObAMAction value)throws ObAMException;
}
```

Class ObAMAuthorizationRule

```
public class ObAMAuthorizationRule {
   public ObAMAuthorizationRule();
   public String getName();
   public String getDescription();
   public boolean getEnabled();
   public boolean getAllowTakesPrecedence();
   public ObAMTimingConditions getTimingConditions();
   public int getNumberOfActions(int actionType)
         throws ObAMObjection;
   public ObAMAction getActionOfType(int actionType,
         int index)throws ObAMException;
   public void addActionOfType(int actionTtype,
         ObAMAction value)throws ObAMException;
   public void removeActionOfType(int actionType,
         ObAMAction value)throws ObAMException;
   public ObAMParameter getSchemeParameter(int index);
   public ObAMAccessConditionsgetAllowAccessConditions();
   public ObAMAccessConditionsgetDenyAccessConditions();
   public String getAuthorizatonScheme();
   public void setName(String value);
   public void setDescription(String value);
   public void setEnabled(boolean value);
   public void setAllowTakesPrecedence(boolean value);
   public void setTimingConditions(
         ObAMTimingConditions value);
   public void setAuthorizationScheme(String value);
   public void addSchemeParameter(
         ObAMParameter value):
```

```
   public void removeSchemeParameter(
         ObAMParameter value);
   public void modifySchemeParameter(
         ObAMParameter value);
   public void setIDFrom(ObAMAuthorizationRule other);
}
```

Class ObAMAuthorizationExpr

```
public class ObAMAuthorizationExpr {
   public static final int ACTION_DUPLICATE = 0;
   public static final int ACTION_IGNORE = 1;
   public static final int ACTION_OVERWRITE = 2;
   public static final int UNDEFINED = 3;
   public ObAMAuthorizationExpr();
   public String getExpression();
   public void setExpression(String value) throws ObAMException;
   public int getDuplicateActionsPolicy();
   public void setDuplicateActionsPolicy(int value)
         throws ObAMException;
   public int getNumberOfActions(int actionType)
         throws ObAMException;
   public ObAMAction getActionOfType(
         int actionType,int index) throws ObAMException;
   public void addActionOfType(int actionType, ObAMAction value)
         throws ObAMException;
   public void removeActionOfType(int actionType,
         ObAMAction value) throws ObAMException;
   public String getName();
   public void setName(String value);
   public void setIDFrom(ObAMObjectWithActions other);
}
```

Class ObAMAuditRule

```
public class ObAMAuditRule {
   public static final int AUTHENTICATION_SUCCESS = 0x01;
   public static final int AUTHENTICATION_FAILURE = 0x02;
   public static final int AUTHORIZATION_SUCCESS  = 0x04;
   public static final int AUTHORIZATION_FAILURE  = 0x08;
   public static final int NUMBER_OF_AUDIT_EVENTS = 4;
   public ObAMAuditRule();
   public int ObAMAuditRule_getNumberOfEvents(
         ObAMAuditRule_t audit);
   public int ObAMAuditRule_getNumberOfAttributes(
         ObAMAuditRule_t audit);
   public int ObAMAuditRule_getEvent(
         ObAMAuditRule_t audit, int index);
   public String ObAMAuditRule_getAttribute(
         ObAMAuditRule_t audit,int index);
   public void addEvent(int value);
```

```
   public void addAttribute(String value);
   public void removeEvent(int value) ;
   public void removeAttribute(String value);
   public void setIDFrom(ObAMAuditRule other);
}
```

Class ObAMAdminRule

```
public class ObAMAdminRule {
   public ObAMAdminRule();
   public int getNumberOfPersons();
   public int getNumberOfGroups();
   public int getNumberOfRoles();
   public int getNumberOfRules();
   public ObAMIdentity_t getPerson(int index);
   public ObAMIdentity_t getGroup(int index);
   public String getRole(int index);
   public String getRule(int index);
   public void addRole(String value);
   public void addPerson(ObAMIdentity value);
   public void addGroup(ObAMIdentity value);
   public void addRule(String value)
   public void removeRole(String value);
   public void removePerson(ObAMIdentity value);
   public void removeGroup(ObAMIdentity value);
   public void removeRule(String value);
   public void setIDFrom(ObAMAdminRule other);
}
```

Class ObAMParameter

```
   public class ObAMParameter {
      public ObAMParameter();
      public String getName();
      public String getValue();
      public void setName(String value);
      public void setValue(String value);
   }
```

Class ObAMPolicy

```
   public class ObAMPolicy {
      public ObAMPolicy();
      public String getName();
      public String getDescription();
      public String getResourceType();
      public String getHostID();
      public String getURLPattern();
      public String getQueryString();
      public int getNumberOfOperations();
```

```
    public int getNumberOfResources();
    public int getNumberOfParameters();
    public int ObAMAuthorizationExpression
         getAuthorizationExpr();
    public void setAuthorizationExpr(
         ObAMAuthorizationExpr value);
    public String getOperation(int index);
    public ObAMResource getResource(int index);
    public ObAMParameter getParameter(int index);
    public ObAMAuthenticationRule
         getAuthenticationRule();
    public ObAMAuditRule getAuditRule();
    public void setName(String value);
    public void setDescription(String value);
    public void setResourceType(String value);
    public void setHostID(String value);
    public void setURLPattern(String value);
    public void setQueryString(String value);
    public void setAuditRule(ObAMAuditRule value);
    public void addOperation(String value);
    public void addResource(ObAMResource value);
    public void addParameter(ObAMParameter value);
    public void removeOperation(String value);
    public void removeResource(ObAMResource value);
    public void removeParameter(Map value);
    public void setIDFrom(ObAMPolicy other);
}
```

Class ObAMPolicyDomain

```
public class ObAMPolicyDomain extends ObListElement {
     public ObAMPolicyDomain()
     public String getName()
     public String getDescription()
     public boolean getEnabled()
     public int getNumberOfResources();
     public int getNumberOfAuthorizationRules();
     public ObAMAuthorizationRule getAuthorizationRule(int index);
     public ObAMAuthorizationExpr getDefaultAuthorizationExpr();
     public int getNumberOfPolicies();
     public ObAMResource getResource(int index);
     public ObAMAuthenticationRule
          getDefaultAuthenticationRule(int index);
     ObAMPolicy getPolicy(int index);
     public ObAMAuditRule getDefaultAuditRule();
     public ObAMAdminRule getDelegateAdminRule();
     public ObAMAdminRule getGrantAdminRule();
     public ObAMAdminRule getBasicAdminRule();
     public void modifyPolicy(ObAMpolicy value):
     public void modifyAuthorizationRule(
          ObAMAuthorizationRule value):
```

```
   public void modifyResource(ObResource value):
   public void setName(String value);
   public void setDescription(String value);
   public void setEnabled(boolean value);
   public void setDefaultAuthenticationRule(
         ObAMAuthenticationRule value);
   public void setDefaultAuditRule(
         ObAMAuditRule value);
   public void setDefaultAuthorizationExpr(
         ObAMAuthorizationExpr value);
   public void setDelegateAdminRule(
         ObAMAdminRule value);
   public void setGrantAdminRule(ObAMAdminRule value);
   public void setBasicAdminRule(ObAMAdminRule value);
   public void addResource(ObAMResource value);
   public void addPolicy(ObAMPolicy value);
   public void addAuthorizationRule(
         ObAMAuthorizationRule value);
   public void removeAuthorizationRule(
         ObAMAuthorizationRule value);
   public void removeResource(ObAMResource value);
   public void removePolicy(ObAMPolicy value);
   public void setIDFrom(ObAMPolicyDomain other);
}
```

Class ObAMAccessTest

```
public class ObAMAccessTest {
   public ObAMAccessTest();
   public String getURL();
   public String getResourceType();
   public String getIPaddress();
   public ObAMDate getDate();
   public ObAMTime getTime();
   int getNumberOfOperations();
   int getNumberOfUsers();
   String getOperation(int index);
   ObAMIdentity getUser(int index);
   public boolean getShowAllowed();
   public boolean getShowDenied();
   public boolean getShowMatchingPolicy();
   public boolean getShowMatchingExpr();
   public boolean getShowDeterminingRules();
   public void setURL(String value);
   public void setResourceType(String value);
   public void addOperation(String value);
   public void setIPaddress(String value);
   public void setDate(ObAMDate value);
   public void setTime(ObAMTime value);
   public void addUser(ObAMIdentity value);
   public void setShowAllowed(boolean value);
   public void setShowDenied(boolean value);
```

```
   public void setShowMatchingPolicy(boolean value);
   public void setShowMatchingExpr(boolean value);
   public void setShowDeterminingRules(boolean value);
}
```

Class ObAMAccessTestResults

```
public class ObAMAccessTestResults {
   public String getPolicyDomain();
   public int getNumberOfResults();
   public ObAMAccessTestResult getResult(int index);
}
```

Class ObAMAccessTestResult(s)

```
public class ObAMAccessTestResult {
   public ObAMIdentity getUser();
   public String getPolicy();
   public boolean getAuthorized();
   public String getExpr();
   public int getNumberOfDeterminingRules();
   public String getDeterminingRule (int index);
   public int getAuthorizationStatus();
}
```

C

Class ObAMResource

```
typedef const void * ObAMResource_t;
   ObAMResource_t ObAMResource_new();
   ObAMResource_t ObAMResource_copy(
         ObAMResource_t resource);
   void ObAMResource_delete(ObAMResource_t *pResource);
   const char *ObAMResource_getResourceType(
         ObAMResource_t resource);
   const char *ObAMResource_getHostID(ObAMResource_t resource);
   const char *ObAMResource_getURLPrefix(
         ObAMResource_t resource);
   const char *ObAMResource_getDescription(
         ObAMResource_t resource);
   void ObAMResource_setResourceType(ObAMResource_t resource,
         const char *value);
   void ObAMResource_setHostID(ObAMResource_t resource,
         const char *value);
   void ObAMResource_setURLPrefix(ObAMResource_t resource,
         const char *value);
   void ObAMResource_setDescription(ObAMResource_t resource,
         const char *value);
```

```
   void ObAMResource_SetIDFrom(ObAMResource_t resource,
        ObAMResource_t other);
```

Class ObAMAccessConditions

```
typedef const void * ObAMAccessConditions_t;
   ObAMAccessConditions_t ObAMAccessConditions_new();
   ObAMAccessConditions_t ObAMAccessConditions_copy(
        ObAMAccessConditions_t access);
   void ObAMAccessConditions_delete(
        ObAMAccessConditions_t *pAccess);
   int ObAMAccessConditions_getNumberOfPersons(
        ObAMAccessConditions_t access);
   int ObAMAccessConditions_getNumberOfGroups(
        ObAMAccessConditions_t access);
   int ObAMAccessConditions_getNumberOfRoles(
        ObAMAccessConditions_t access);
   int ObAMAccessConditions_getNumberOfRules(
        ObAMAccessConditions_t access);
   int ObAMAccessConditions_getNumberOfIPaddresses(
        ObAMAccessConditions_t access);
   ObAMIdentity_t ObAMAccessConditions_getPerson(
        ObAMAccessConditions_t access, int index);
   ObAMIdentity_t ObAMAccessConditions_getGroup(
        ObAMAccessConditions_t access, int index);
   const char *ObAMAccessConditions_getRole(
        ObAMAccessConditions_t access, int index);
   const char *ObAMAccessConditions_getRule(
        ObAMAccessConditions_t access, int index);
   const char *ObAMAccessConditions_getIPaddress(
        ObAMAccessConditions_t access, int index);
   void ObAMAccessConditions_addRole(
        ObAMAccessConditions_t access,const char *value);
   void ObAMAccessConditions_addPerson(
        ObAMAccessConditions_t access, ObAMIdentity_t value);
   void ObAMAccessConditions_addGroup(
        ObAMAccessConditions_t access, ObAMIdentity_t value);
   void ObAMAccessConditions_addRule(
        ObAMAccessConditions_t access, const char *value);
   void ObAMAccessConditions_addIPAddress(
        ObAMAccessConditions_t access, const char *value);
```

Class ObAMDate

```
enum ObAMDate_DaysOfWeek {
   ObAMDate_SUNDAY    = 1,
   ObAMDate_MONDAY    = 2,
   ObAMDate_TUESDAY   = 3,
   ObAMDate_WEDNESDAY = 4,
   ObAMDate_THURSDAY  = 5,
   ObAMDate_FRIDAY    = 6,
```

```
   ObAMDate_SATURDAY  = 7};
enum ObAMDate_Months {
   ObAMDate_JANUARY   = 0,
   ObAMDate_FEBRUARY  = 1,
   ObAMDate_MARCH     = 2,
   ObAMDate_APRIL     = 3,
   ObAMDate_MAY       = 4,
   ObAMDate_JUNE      = 5,
   ObAMDate_JULY      = 6,
   ObAMDate_AUGUST    = 7,
   ObAMDate_SEPTEMBER = 8,
   ObAMDate_OCTOBER   = 9,
   ObAMDate_NOVEMBER  = 10,
   ObAMDate_DECEMBER  = 11};
typedef const void * ObAMDate_t;
   ObAMDate_t ObAMDate_new();
   ObAMDate_t ObAMDate_copy(ObAMDate_t date);
   void ObAMDate_delete(ObAMDate_t date);
   int ObAMDate_getYear(ObAMDate_t date);
   int ObAMDate_getMonth(ObAMDate_t date);
   int ObAMDate_getDay(ObAMDate_t date);
   void ObAMDate_set(
        ObAMDate_t date, int year, int month, int day);
```

Class ObAMTime

```
typedef const void * ObAMTime_t;
   ObAMTime_t ObAMTime_new();
   ObAMTime_t ObAMTime_copy(ObAMTime_t time);
   void ObAMTime_delete(ObAMTime_t time);
   int ObAMTime_getMonth(ObAMTime_t time);
   int ObAMTime_getDay(ObAMTime_t time);
   int ObAMTime_getYear(ObAMTime_t time);
   void ObAMTime_set(ObAMTime_t time, int hours, int minutes,
        int seconds);
```

Class ObAMTimingConditions

```
enum ObAMTimingConditions_RelativeTo {
   ObAMTimingConditions_UNDEFINED,
   ObAMTimingConditions_GMT,
   ObAMTimingConditions_LOCAL_Time
};
typedef const char * ObAMTimingConditions_t;
   ObAMTimingConditions_t ObAMTimingConditions_new();
   ObAMTimingConditions_t ObAMTimingConditions_copy(
        ObAMTimingConditions_t timing);
   void ObAMTimingConditions_delete(
        ObAMTimingConditions_t *pTiming);
ObAMTimingConditions_RelativeTo ObAMTimingConditions_getRelativeTo(
        ObAMTimingConditions_t timing);
```

```
ObAMDate_t ObAMTimingConditions_getStartDate(
      ObAMTimingConditions_t timing);
ObAMTime_t ObAMTimingConditions_getStartTime(
      ObAMTimingConditions_t timing);
ObAMDate_t ObAMTimingConditions_getEndDate(
      ObAMTimingConditions_t timing);
ObAMTime_t ObAMTimingConditions_getEndTime(
      ObAMTimingConditions_t timing);
int ObAMTimingConditions_getNumberOfMonths(
      ObAMTimingConditions_t timing);
int ObAMTimingConditions_getNumberOfDaysOfMonth(
      ObAMTimingConditions_t timing);
int ObAMTimingConditions_getNumberOfDaysOfWeek(
      ObAMTimingConditions_t timing);
int ObAMTimingConditions_getMonth(
      ObAMTimingConditions_t timing,int index);
int ObAMTimingConditions_getDayOfMonth(
      ObAMTimingConditions_t timing,int index);
int ObAMTimingConditions_getDayOfWeek(
      ObAMTimingConditions_t timing,int index);
void ObAMTimingConditions_setRelativeTo(
      ObAMTimingConditions_t timing,
      ObAMTimingConditions_RelativeTo value);
void ObAMTimingConditions_setStartDate(
      ObAMTimingConditions_t timing,ObAMDate_t value);
void ObAMTimingConditions_setStartTIME(
      ObAMTimingConditions_t timing, ObAMTime_t value);
void ObAMTimingConditions_setEndDate(
      ObAMTimingConditions_t timing, ObAMDate_t value);
void ObAMTimingConditions_setEndTime(
      ObAMTimingConditions_t timing, ObAMTime_t value);
void ObAMTimingConditions_addMonth(
      ObAMTimingConditions_t timing, int value);
void ObAMTimingConditions_addDayOfMonth(
      ObAMTimingConditions_t timing, int value);
void ObAMTimingConditions_addDayOfWeek(
      ObAMTimingConditions_t timing, int value);
void ObAMTimingConditions_removeMonth(
      ObAMTimingConditions_t timing, int value);
void ObAMTimingConditions_removeDayOfMonth(
      ObAMTimingConditions_t timing, int value);
void ObAMTimingConditions_removeDayOfWeek(
      ObAMTimingConditions_t timing, int value);
void ObAMTimingConditions_setIDFrom(
      ObAMTimingConditions_t timing,
      ObAMTimingConditions_t other);
```

Class ObAMIdentity

```
typedef const void * ObAMIdentity_t;
```

```
   ObAMIdentity_t ObAMIdentity_new();
   ObAMIdentity_t ObAMIdentity_copy(
         ObAMIdentity_t identity);
   void ObAMIdentity_delete(ObAMIdentity_t *pIdentity);
   const char *ObAMIdentity_getUID(
         ObAMIdentity_t identity);
   const char *ObAMIdentity_getName(ObAMIdentity_t identity);
   const char *ObAMIdentity_getLoginID(
         ObAMIdentity_t identity);
   void ObAMIdentity_setUID(ObAMIdentity_t identity,
         const char *value);
   void ObAMIdentity_setName(ObAMIdentity_t identity,
         const char *value);
   void ObAMIdentity_setLoginID(ObAMIdentity_t identity,
         const char *value);
```

Class ObAMAction

```
enum ObAMAction_ValueType {
   ObAMAction_UNDEFINED,
   ObAMAction_FIXEDVALUE,
   ObAMAction_ATTRIBUTE;
};
typedef const void *ObAMAction_t;
   ObAMAction_t ObAMAction_new();
   ObAMAction_t ObAMAction_copy(ObAMAction_t action);
   void ObAMAction_delete(ObAMAction_t *pAction);
   const char *ObAMAction_getType(ObAMAction_t action);
   const char *ObAMAction_getName(ObAMAction_t action);
   const char *ObAMAction_getValue(ObAMAction_t action);
   ObAMAction_ValueType ObAMAction_getValueType(
         ObAMAction_t action);
   void ObAMAction_setType(
         ObAMAction_t action, const char *value);
   void ObAMAction_setName(
         ObAMAction_t action, const char *value);
   void ObAMAction_setValue(ObAMAction_t action,
         const char *value);
   void ObAMAction_setValueType(ObAMAction_t action,
         ObAMAction_ValueType value);
   void ObAMAction_setIDFrom(ObAMAction_t action,
         ObAMAction_t other);
```

Class ObAMObjectWithActions

```
enum ObAMObjectWithActions_ActionType{
   ObAMObjectWithActions_SUCCESS,
   ObAMObjectWithActions_FAILURE,
   ObAMObjectWithActions_INCONCLUSIVE
};
```

Class ObAMAuthenticationRule

```
typedef const void * ObAMAuthenticationRule_t;
   ObAMAuthenticationRule_t ObAMAuthenticationRule_new();
   ObAMAuthenticationRule_t ObAMAuthenticationRule_copy(
         ObAMAuthenticationRule_t authn);
   void ObAMAuthenticationRule_delete(
         ObAMAuthenticationRule_t *pAuthn);
   const char *ObAMAuthenticationRule_getName(
         ObAMAuthenticationRule_t authn);
   const char *ObAMAuthenticationRule_getDescription(
         ObAMAuthenticationRule_t authn);
   const char *ObAMAuthenticationRule_getScheme(
         ObAMAuthenticationRule_t authn);
   int ObAMAuthenticationRule_getNumberOfActions(
         ObAMObjectWithActions_ActionType type,
         ObAMAuthenticationRule_t authn);
   ObAMAction_t ObAMAuthenticationRule_getActionOfType(
         ObAMObjectWithAction_ActionType type,
         ObAMAuthenticationRule_t authn, int index);
   void ObAMAuthenticationRule_addActionOfType(
         ObAMObjectWithActions_ActionType type,
         ObAMAuthationRule_t authn,ObAMAction_t value);
   void ObAMAuthnenticationRule_removeActionOfType(
         ObAMObjectWithActions_ActionType type,
         ObAMAuthenticationRule_t authn, ObAMAction_t value);
   void ObAMAuthenticationRule_setName(
         ObAMAuthenticationRule_t authn, const char *value);
   void ObAMAuthenticationRule_setDescription(
         ObAMAuthenticationRule_t authn, const char *value);
   void ObAMAuthenticationRule_setScheme(
         ObAMAuthenticationRule_t authn, const char *value);
   void ObAMAuthenticationRule_setIDFrom(
         ObAMAuthenticationRule_t authn,
         ObAMAuthenticationRule_T other);
```

Class ObAMAuthorizationRule

```
typedef const void * ObAMAuthorizationRule_t;
   ObAMAuthorizationRule_t ObAMAuthorizationRule_new();
   ObAMAuthorizationRule_t ObAMAuthorizationRule_copy(
         ObAMAuthorizationRule_t authz);
   void ObAMAuthorizationRule_delete(
         ObAMAuthorizationRule_t *pAuthz);
   const char *ObAMAuthorizationRule_getName(
         ObAMAuthorizationRule_t authz);
   const char *ObAMAuthorizationRule_getDescription(
         ObAMAuthorizationRule_t authz);
   int ObAMAuthorizationRule_getEnabled(
         ObAMAuthorizationRule_t authz);
   int ObAMAuthorizationRule_getAllowTakesPrecedence(
         ObAMAuthorizationRule_t authz);
```

```
ObAMTimingConditions_t
      ObAMAuthorizationRule_getTimingConditions(
      ObAMAuthorizationRule_t authz);
   int ObAMAuthorizationRule_getNumberOfSchemeParameters(
      ObAMAuthorizationRule_t authz);
   ObAMAuthorizationRule_getSchemeParameter(
         ObAMAuthorizationRule_t authz, int index);
   int ObAMAuthorizationRule_getNumberOfActions(
         ObAMObjectWithActions_ActionType type,
         ObAMAuthorizationRule_t authz);
   ObAMAction_t ObAMAuthorizationRule_getActionOfType(
         ObAMObjectWithActions_ActionType type,
         ObAMAuthorizationRule_t authz, int index);
ObAMAccessConditions_t
      ObAMAuthorizationRule_getAllowAccessConditions(
      ObAMAuthorizationRule_t authz);
ObAMAccessConditions_t
      ObAMAuthorizationRule_getDenyAccessConditions(
      ObAMAuthorizationRule_t authz);
const char *ObAMAuthorizationRule_getAuthorizationScheme(
      ObAMAuthorizationRule_t authz);
   void ObAMAuthorizationRule_setName(
         ObAMAuthorizationRule_t authz, const char *value);
   void ObAMAuthorizationRule_setDescription(
         ObAMAuthorizationRule_t authz, const char *value);
   void ObAMAuthorizationRule_setEnabled(
         ObAMAuthorizationRule_t authz, int value);
   void ObAMAuthorizationRule_setAllowTakesPrecedence(
         ObAMAuthorizationRule_t authz, int value);
   void ObAMAuthorizationRule_setTimingConditions(
         ObAMAuthorizationRule_t authz,
         ObAMTimingConditions_t value);
   void ObAMAuthorizationRule_setAllowAccessConditions(
         ObAMAuthorizationRule_t authz,
         ObAMAccessConditions_t value);
   void ObAMAuthorizationRule_setDenyAccessConditions(
         ObAMAuthorizationRule_t authz,
         ObAMAccessConditions_t value);
   void ObAMAuthorizationRule_setAuthorizationScheme(
         ObAMAuthorizationRule_t authz,
   void ObAMAuthorizationRule_addSchemeParameter(
         ObAMAuthorizationRule_t authz, ObAMParameter_t value);
   void ObAMAuthorizationRule_removeSchemeParamter(
         ObAMAuthorizationRule_t authz, ObAMParameter_t value);
   void ObAMAuthorizationRule_modifySchemeParameter(
         ObAMAuthorizationRule_t authz, ObAMParameter_t value);
   void ObAMAuthorizationRule_setIDFrom(
         ObAMAuthorizationRule_t authz,
         ObAMAuthorizationRule_t other);
   int ObAMAuthorizationRule_getNumberOfActions(
         ObAMObjectWithActions_ActionType type,
         ObAMAuthorizationRule_t authz);
ObAMAction_t ObAMAuthorizationRule_getActionOfType(
```

```
        ObAMObjectWithActions_ActionType type,
        ObAMAuthorizationRule_t authz, int index);
   void ObAMAuthorizationRule_addActionOfType(
        ObAMObjectWithActions_ActionType type,
        ObAMAuthorizationRule_t authz, ObAMAction_t value);
```

Class ObAMAuthorizationExpr

```
enum ObAMAuthorizationExpr_DuplicateActionsPolicy {
     ObAMAuthorizationExpr_ACTION_DUPLICATE,
     ObAMAAuthorizationExpr_ACTION_DUPLICATE,
     ObAMAuthorizationExpr_ACTION_IGNORE,
     ObAMAuthorizationExpr_ACTION_OVERWRITE,
   ObAMAuthorizationExpr_UNDEFINED
};
typedef const void * ObAMAuthorizationExpr_t;
   ObAMAuthorizationExpr_t ObAMAuthorizationExpr_new();
   ObAMAuthorizationExpr_t ObAMAuthorizationExpr_copy(
        ObAMAuthorizationExpr_t authz);
   void ObAMAuthorizationExpr_delete(
        ObAMAuthorizationExpr_t *pAuthz);
   const char *ObAMAuthorizationExpr_getExpr(
        ObAMAuthorizationExpr_t authz);
   int ObAMAuthorizationExpr_getDuplicateActionsPolicy(
        ObAMAuthorizationExpr_t authz);
   int ObAMAuthorizationExpr_getNumberOfActions(
        ObAMObjectWithActions_ActionType type,
        ObAMAuthorizationExpr_t authz);
   ObAMAction_t ObAMAuthorizationExpr_getActionOfType(
        ObAMObjectWithActions_ActionType type,
        ObAMAuthorizationExpr_t authz, int index);
   void ObAMAuthorizationExpr_setExpr(
        ObAMAuthorizationExpr_t authz, const char *value);
   void ObAMAuthorizationExpr_setDuplicateActionsPolicy(
        ObAMAuthorizationExpr_t authz,
        ObAMAuthorizationExpr_DuplicateActionsPolicy value);
   void ObAMAuthorizationExpr_addActionOfType(
        ObAMObjectWithActions_ActionType type,
        ObAMAuthorizationExpr_t authz,ObAMAction_t value);
   void ObAMAuthorizationExpr_removeActionOfType(
        ObAMObjectWithActions_ActionType type,
        ObAMAuthorizationExpr_t authz, ObAMAction_t value);
```

Class ObAMAuditRule

```
enum ObAMAuditRule_EventType {
   ObAMAuditRule_AUTHENTICATION_SUCCESS = 0x01,
   ObAMAuditRule_AUTHENTICATION_FAILURE = 0x02,
   ObAMAuditRule_AUTHORIZATION_SUCCESS  = 0x04,
   ObAMAuditRule_AUTHORIZATION_FAILURE  = 0x08};
```

```
typedef const void * ObAMAuditRule_t;
typedef unsigned int ObAMAuditRule_EventTypeMask;
   ObAMAuditRule_t ObAMAuditRule_new();
   ObAMAuditRule_t ObAMAuditRule_copy(ObAMAuditRule_t audit);
   void ObAMAuditRule_delete(ObAMAuditRule_t *pAudit);
   int ObAMAuditRule_getNumberOfEvents(ObAMAuditRule_t audit);
   int ObAMAuditRule_getNumberOfAttributes(ObAMAuditRule_t audit);
   ObAMAuditRule_EventType ObAMAuditRule_getEvent(
         ObAMAuditRule_t audit, int index);
   const char *ObAMAuditRule_getAttribute(ObAMAuditRule_t audit,
         int index);
   void ObAMAuditRule_addEvent(ObAMAuditRule_t audit,
         ObAMAuditRule_EventType value);
   void ObAMAuditRule_addAttribute(ObAMAuditRule_t audit,
         const char *value);
   void ObAMAuditRule_removeEvent(ObAMAuditRule_t audit,
         ObAMAuditRule_EventType value);
   void ObAMAuditRule_removeAttribute(ObAMAuditRule_t audit,
         const char *value);
   void ObAMAuditRule_setIDFrom(ObAMAuditRule_t audit,
         ObAMAuditRule_t other);
```

Class ObAMAdminRule

```
   typedef const void * ObAMAdminRule_t;
      ObAMAdminRule_t ObAMAdminRule_new();
      ObAMAdminRule_t ObAMAdminRule_copy(
            ObAMAdminRule_t admin);
      void ObAMAdminRule_delete(ObAMAdminRule_t *pAdmin);
      int ObAMAdminRule_getNumberOfPersons(
            ObAMAdminRule_t admin);
      int ObAMAdminRule_getNumberOfGroups(
            ObAMAdminRule_t admin);
      int ObAMAdminRule_getNumberOfRoles(
            ObAMAdminRule_t admin);
      int ObAMAdminRule_getNumberOfRules(
            ObAMAdminRule_t admin);
      ObAMIdentity_t ObAMAdminRule_getPerson(
            ObAMAdminRule_t admin, int index);
      ObAMIdentity_t ObAMAdminRule_getGroup(
            ObAMAdminRule_t admin, int index);
      const char *ObAMAdminRule_getRole(
            ObAMAdminRule_t admin, int index);
      const char *ObAMAdminRule_getRule(
            ObAMAdminRule_t admin,int index);
      void ObAMAdminRule_addRole(ObAMAdminRule_t admin,
            const char *value);
      void ObAMAdminRule_addPerson(
            ObAMAdminRule_t admin,ObAMIdentity_t value);
      void ObAMAdminRule_addGroup(ObAMAdminRule_t admin,
            ObAMIdentity_t value);
      void ObAMAdminRule_addRule(ObAMAdminRule_t admin,
```

```
            const char *value);
      void ObAMAdminRule_removeRole(ObAMAdminRule_t admin,
            const char *value);
      void ObAMAdminRule_removePerson(ObAMAdminRule_t admin,
            ObAMIdentity_t value);
      void ObAMAdminRule_removeGroup(ObAMAdminRule_t admin,
            ObAMIdentity_t value);
void ObAMAdminRule_removeRule(ObAMAdminRule_t admin,
            const char *value);
void ObAMAdminRule_setIDFrom(ObAMAdminRule_t admin,
            ObAMAdminRule_t other);
```

Class ObAMParameter

```
   typedef const void * ObAMParameter_t;
      ObAMParameter_t ObAMParameter_new();
      ObAMParameter_t ObAMParameter_copy(ObAMParameter_t param);
      void ObAMParameter_delete(ObAMParameter_t *pParam);
      const char *ObAMParameter_getName(ObAMParameter_t param);
      const char *ObAMParameter_getValue(ObAMParameter_t param);
      void ObAMParameter_setName(ObAMParameter_t param,
            const char *value);
      void ObAMParameter_setValue(ObAMParameter_t param,
            const char *value);
```

Class ObAMPolicy

```
   typedef const void * ObAMPolicy_t;
      ObAMPolicy_t ObAMPolicy_new();
      ObAMPolicy_t ObAMPolicy_copy(ObAMPolicy_t policy);
      void ObAMPolicy_delete(ObAMPolicy_t *pPolicy);
      const char *ObAMPolicy_getName(ObAMPolicy_t policy);
      const char *ObAMPolicy_getDescription(ObAMPolicy_t policy);
      const char *ObAMPolicy_getResourceType(ObAMPolicy_t policy);
      const char *ObAMPolicy_getHostID(ObAMPolicy_t policy);
      const char *ObAMPolicy_getURLPattern(ObAMPolicy_t policy);
      const char *ObAMPolicy_getQueryString(ObAMPolicy_t policy);
      int ObAMPolicy_getNumberOfOperations(ObAMPolicy_t policy);
      int ObAMPolicy_getNumberOfResources(ObAMPolicy_t policy);
      int ObAMPolicy_getNumberOfParameters(ObAMPolicy_t policy);
      const char *ObAMPolicy_getOperation(ObAMPolicy_t policy,
            int index);
      ObAMResource_t ObAMPolicy_getResource(ObAMPolicy_t policy,
            int index);
      ObAMParameter_t ObAMPolicy_getParameter(ObAMPolicy_t policy,
            int index);
      ObAMAuthenticationRule_t
            ObAMPolicy_getAuthenticationRule(ObAMPolicy_t policy);
      ObAMAuditRule_t ObAMPolicy_getAuditRule(ObAMPolicy_t policy);
      void ObAMPolicy_setName(
            ObAMPolicy_t policy,  const char *value);
```

```
void ObAMPolicy_setDescription(ObAMPolicy_t policy
     const char *value);
void ObAMPolicy_setResourceType(ObAMPolicy_t policy,
     const char *value);
void ObAMPolicy_setHostID(ObAMPolicy_t policy,
     const char *value);
void ObAMPolicy_setURLPattern(ObAMPolicy_t policy,
     const char *value);
void ObAMPolicy_setQueryString(ObAMPolicy_t policy,
     const char *value);
void ObAMPolicy_setAuthenticationRule(ObAMPolicy_t policy,
     ObAMAuthenticationRule_t value);
void ObAMPolicy_setAuditRule(ObAMPolicy_t policy,
     ObAMAuditRule_t value);
void ObAMPolicy_addOperation(ObAMPolicy_t policy,
     const char *value);
void ObAMPolicy_addResource(ObAMPolicy_t policy,
     ObAMResource_t value);
void ObAMPolicy_addParameter(ObAMPolicy_t policy,
     ObAMParameter_t value);
void ObAMPolicy_removeOperation(ObAMPolicy_t policy,
     const char *value);
void ObAMPolicy_removeResource(ObAMPolicy_t policy,
     ObAMResource_t value);
void ObAMPolicy_removeParameter(ObAMPolicy_t policy,
     ObAMParameter_t value);
void ObAMPolicy_setIDFrom(ObAMPolicy_t policy,
     ObAMPolicy_t other);
```

Class ObAMPolicyDomain

```
typedef const void * ObAMPolicyDomain_t;
  ObAMPolicyDomain_t ObAMPolicyDomain_new();
  ObAMPolicyDomain_t ObAMPolicyDomain_copy(
       ObAMPolicyDomain_t domain);
  void ObAMPolicyDomain_delete(
       ObAMPolicyDomain_t *pDomain);
  const char *ObAMPolicyDomain_getName(
       ObAMPolicyDomain_t domain);
  const char *ObAMPolicyDomain_getDescription(
       ObAMPolicyDomain_t domain);
  int ObAMPolicyDomain_getEnabled(
       ObAMPolicyDomain_t domain);
  int ObAMPolicyDomain_getNumberOfResources(
       ObAMPolicyDomain_t domain);
  int ObAMPolicyDomain_getNumberOfPolicies(
       ObAMPolicyDomain_t domain);
  ObAMResource_t ObAMPolicyDomain_getResource(
       ObAMPolicyDomain_t domain, int index);
  ObAMPolicy_t ObAMPolicyDomain_getPolicy(
       ObAMPolicyDomain_t domain, int index);
  ObAMAuditRule_t ObAMPolicyDomain_getDefaultAuditRule(
```

```
      ObAMPolicyDomain_t domain);
ObAMAuthorizationRule_t
      ObAMAPolicyDomain_getAuthorizationRule(
      ObAMPolicyDomain_t domain, int index);
ObAMAuthorizationExpr_t
      ObAMPolicyDomain_getDefaultAuthorizationExpr(
      ObAMPolicyDomain_t domain);
void ObAMPolicyDomain_setDefaultAuthorizationExpr(
      ObAMPolicyDomain_t domain,
      ObAMAuthorizationExpr_t value);
ObAMAdminRule_t ObAMPolicyDomain_getDelegateAdminRule(
      ObAMPolicyDomain_t domain);
ObAMAdminRule_t ObAMPolicyDomain_getGrantAdminRule(
      ObAMPolicyDomain_t domain);
ObAMAdminRule_t ObAMPolicyDomain_getBasicAdminRule(
      ObAMPolicyDomain_t domain);
void ObAMPolicyDomain_setName(ObAMPolicyDomain_t domain,
      const char *value);
void ObAMPolicyDomain_setDescription(
      ObAMPolicyDomain_t domain, const char *value);
void ObAMPolicyDomain_setEnabled(ObAMPolicyDomain_t domain,
      bool value);
void ObAMPolicyDomain_setDefaultAuthenticationRule(
      ObAMPolicyDomain_t domain,
      ObAMAuthenticationRule_t value);
void ObAMPolicyDomain_setDefaultAuditRule(
      ObAMPolicyDomain_t domain, ObAMAuditRule_t value);
void ObAMPolicyDomain_setDelegateAdminRule(
      ObAMPolicyDomain_t domain, ObAMAdminRule_t value);
void ObAMPolicyDomain_setGrantAdminRule(
      ObAMPolicyDomain_t domain, ObAMAdminRule_t value);
void ObAMPolicyDomain_setBasicAdminRule(
      ObAMPolicyDomain_t domain, ObAMAdminRule_t value);
void ObAMPolicyDomain_addResource(ObAMPolicyDomain_t domain,
      ObAMResource_t value);
void ObAMPolicyDomain_addPolicy(ObAMPolicyDomain_t domain,
      ObAMPolicy_t value);
void ObAMPolicyDomain_addAuthorizationRule(
      ObAMPolicyDomain_t domain,
      ObAMAuthorizationRule_t value);
void ObAMPolicyDomain_modifyAuthorizationRule(
      ObAMPolicyDomain_t domain,
      ObAMAuthorizationRule_t value);
void ObAMPolicyDomain_removeAuthorizationRule(
      ObAMPolicyDomain_t domain,
      ObAMAuthorizationRule_t value);
void ObAMPolicyDomain_removeResource(
      ObAMPolicyDomain_t domain,ObAMResource_t value);
void ObAMPolicyDomain_removePolicy(ObAMPolicyDomain_t domain,
      ObAMPolicy_t value);
```

Class ObAMAccessTest

```
typedef const void * ObAMAccessTest_t;
   ObAMAccessTest_t ObAMAccessTest_new();
   ObAMAccessTest_t ObAMAccessTest_copy(ObAMAccessTest_t test);
   void ObAMAccessTest_delete(ObAMAccessTest_t *pTest);
   const char *ObAMAccessTest_getURL(ObAMAccessTest_t test);
   const char *ObAMAccessTest_getResourceType(
         ObAMAccessTest_t test);
   const char *ObAMAccessTest_getIPaddress(
         ObAMAccessTest_t test);
   ObAMDate_t ObAMAccessTest_getDate(
         ObAMAccessTest_t test);
   ObAMTime_t ObAMAccessTest_getTime(
         ObAMAccessTest_t test);
   int ObAMAccessTest_getNumberOfOperations(
         ObAMAccessTest_t test);
   int ObAMAccessTest_getNumberOfUsers(
         ObAMAccessTest_t test);
   const char *ObAMAccessTest_getOperation(
         ObAMAccessTest_t test, int index);
   ObAMIdentity_t ObAMAccessTest_getUser(ObAMAccessTest_t test,
         int index);
   int ObAMAccessTest_getShowAllowed(ObAMAccessTest_t test);
   int ObAMAccessTest_getShowDenied(ObAMAccessTest_t test);
   int ObAMAccessTest_getShowMatchingPolicy(
         ObAMAccessTest_t test);
   void ObAMAccessTest_setURL(ObAMAccessTest_t test,
         const char *value);
   void ObAMAccessTest_setResourceType(ObAMAccessTest_t test,
         const char *value);
   void ObAMAccessTest_addOperation(ObAMAccessTest_t test,
         const char *value);
   void ObAMAccessTest_setIPaddress(ObAMAccessTest_t test,
         const char *value);
   void ObAMAccessTest_setDate(ObAMAccessTest_t test,
         ObAMDate_t value);
   void ObAMAccessTest_setTime(ObAMAccessTest_t test,
         ObAMTime_t value);
   void ObAMAccessTest_addUser(ObAMAccessTest_t test,
         ObAMIdentity_t value);
   void ObAMAccessTest_setShowAllowed(ObAMAccessTest_t test,
         int value);
   void ObAMAccessTest_setShowDenied(ObAMAccessTest_t test,
         int value);
   void ObAMAccessTest_setShowMatchingPolicy(
         ObAMAccessTest_t test, int value);
```

Class ObAMAccessTestResults

```
typedef const void * ObAMAccessTestResults_t;
   void ObAMAccessTestResults_delete(
```

```
        ObAMAccessTestResults_t results);
    const char *ObAMAccessTestResults_getPolicyDomain(
        ObAMAccessTestResults_t results);
    int ObAMAccessTestResults_getNumberOfResults(
        ObAMAccessTestResults_t results);
    ObAMAccessTestResult_t ObAMAccessTestResults_getResult(
        ObAMAccessTestResults_t results, int index);results);
```

Class ObAMAccessTestResult(s)

```
typedef const void * ObAMAccessTestResult_t;
  ObAMIdentity_t ObAMAccessTestResult_getUser(
        ObAMAccessTestResult_t result);
const char *ObAMAccessTestResult_getPolicy(
        ObAMAccessTestResult_t result);
int ObAMAccessTestResult_getAuthorized(
        ObAMAccessTestResult_t result);
const char *ObAMAccessTestResult_getExpr(
        ObAMAccessTestResult_t result);
int ObAMAccecssTestResult_getNumberOfDeterminingRules(
        ObAMAccessTestResult_t result);
const char *ObAMAccessTestResult_getDeterminingRule(
        ObAMAccessTestResult_t result, int index);
int ObAMAccessTestResult_getAuthorizationStatus(
        ObAMAccessTestResult_t result);
```

Managed Code

Class ObAMResourceMgd

```
public __gc class ObAMResourceMgd {
     public:
     ObAMResourceMgd();
     _property System::String *get_ResourceType();
     _property System::String *get_HostID();
     _property System::String *get_URLPrefix();
     _property System::String *get_Description();
     _property void set_IDFrom(ObAMResourceMgd *other);
     _property void set_ResourceType(System::String *value);
     _property void set_HostID(System::String *value);
     _property void set_URLPrefix(System::String *value);
     _property void set_Description(System::String *value);
};
```

Class ObAMAccessConditionsMgd

```
public __gc class ObAMAccessConditionsMgd {
```

```
   public:
      ObAMAccessConditionsMgd();
      _property int get_NumberOfPersons();
      _property int get_NumberOfGroups();
      _property int get_NumberOfRoles();
      _property int get_NumberOfRules();
      _property int get_NumberOfIPaddresses();
      ObAMIdentityMgd *getPerson (int index);
      ObAMIdentityMgd *getGroup (int index);
      System::String *getRole(int index);
      System::String *getRule(int index);
      System::String *getIPaddress(int index);
      _property void set_IDFrom(ObAMAccessConditionsMgd *access);
      _property void set_AddRole(System::String *value);
      _property void set_AddPerson(ObAMIdentityMgd *value);
      _property void set_AddGroup(ObAMIdentityMgd *value);
      _property void set_AddRule(System::String *value);
      _property void set_AddIPaddress(System::String *value);
      _property void set_RemoveRole(System::String *value);
      _property void set_RemoveGroup(ObAMIdentityMgd *value);
      _property void set_RemoveRule(System::String *value);
      _property void set_RemoveIPaddress(System::String *value);
};
```

Class ObAMDateMgd

```
public __gc class ObAMDateMgd {
   public:
      ObAMDATEmGD();
      _property int get_Year();
      _property int get_Month();
      _property int get_Day();
      void set(int year, int month, int day_);
};
```

Class ObAMDate_MonthsMgd

```
public __gc class ObAMDateMgd {
   public:
      ObAMDate_MonthsMgd();
      void setJanuary();
      void setFebruary();
      void setMarch();
      void setApril();
      void setMay();
      void setJune();
      void setJuly();
      void setSeptember();
      void setOctober();
      void setNovember();
      void setDecember();
```

```
      void setOctober();
      void setNovember();
      void setDecember();
};
```

Class ObAMDate_DaysOfWeekMgd

```
public __gc class ObAMDate_DaysOfWeekMgd {
   public:
      ObAMDate_DaysOfWeekMgd();
      void setSunday();
      void setMonday();
      void setTuesday();
      void setWednesday();
      void setThursday();
      void setFriday();
      void setSaturday();
};
```

Class ObAMTimeMgd

```
public __gc class ObAMTimeMgd {
   public:
      ObAmTimeMgd();
      __property int get_Hours();
      __property int getMinutes();
      __property int get_Seconds();
      void set(int hours, int minutes, int seconds);
};
```

Class ObAMTimingConditionsMgd

```
public __gc class ObAMTimingConditions_RelativeToMgd {
   public:
      ObAMTimingConditionsMgd();
      __property ObAMDateMgd *get_StartDate();
      __property ObAMTimingConditions_RelativeToMgd
           *get_RelativeTo();
      __property ObAMTimeMgd *get_StartTime();
      __property ObAMTimeMgd *get_EndDate();
      __property ObAMTimeMgd *get_EndTime();
      __property int get_NumberOfMonths();
      __property int get_NumberOfDaysOfMonth();
      __property get_NumberofDaysOfWeek();
      int getMonth(int index);
```

```
        int getDayOfMonth(int index);
        int getDayOfWeek(int index);
        __property void set_IDFrom(ObAMTimingConditionsMgd *other);
        __property void set_RelativeTo(
             ObAMTimingConditions_RelativeToMgd *value);
        __property void set_StartDate(ObAMDateMgd *date);
        __property void set_EndTime(ObAMTimeMgd *time);
        __property void set_EndDate(ObAMDateMgd *date);
        __property void set_StartTime(ObAMDateMgd *time);
        __property void set_AddMonth(ObAMDate_MonthsMgd *value);
        __property void set_AddDayOfMonth(int value);
        __property void set_AddDayOfWeek(
             ObAMDate_DaysOfWeekMgd *value);
        __property void set_RemoveMonth(ObAMDate_MonthsMgd *value);
        __property void set_RemoveDayOfMonth(int value);
        __property void set_RemoveDayOfWeek(
             ObAMDate_DaysOfWeekMgd *value);
};
```

Class ObAMIdentityMgd

```
public __gc class ObAMIdentityMgd {
   public: ObAMIdentityMgd();
   __property System::String *get_UID();
   __property System::String *get_Name();
   __property System::String *get_LoginID();
   __property ObAMIdentity *get_UnmanageIdentity();
   __property void set_UID(System::String *value);
   __property void set_Name(System::String *value);
   __property void set_LoginID(System::String *value);
};
```

Class ObAMActionTypeMgd

```
public __gc class ObAMActionTypeMgd {
   public:
      void setSuccess();
      void setFailure();
      void setInconclusive();
};
```

Class ObAMActionMgd

```
public __gc class ObAMActionMgd {
   public:
      ObAMActionMgd();
      __property System::String *get_Type();
      __property System::String *get_Name();
      __property System::String *get_value();
      __property ObAMAction_ValueTypeMgd *get_ValueType();
      __property void set_IDFrom(ObAMActionMgd *other);
      __property void set_Type(System::String *value);
```

```
      __property void set_Name(System::String *value);
      __property void set_Value(System::String *value);
};
```

Class ObAMAction_ValueTypeMgd

```
public __gc class ObAMAction_ValueTypeMgd {
   public:
      ObAMAction_ValueTypeMgd();
      __property bool get_isUndefined();
      __property bool get_isFixedValue();
      __property bool get_isAttribute();
      __property ObAMAction_ValueType get_Value();
      __property void set_Value(ObAMAction_ValueType value);
      void setUndefined();
      void setFixedValue();
      void setAttribute();
};
```

Class ObAMAuthenticationRuleMgd

```
public __gc class ObAMAuthenticationMgd {
   public:
      ObAMAuthenticationRuleMgd();
      __property System::String();
      __property System::String *get_Description();
      __property System::String *get_Scheme();
      int getNumberOfActions(ObAMActionTypeMgd *action);
      ObAMActionMgd *getActionOfType(
            ObAMActionTypeMgd *type, int index);
      __property void set_IDFrom(ObAMAuthenticationRuleMgd *other);
      __property void set_Name(System::String *value);
      __property void set_Description(System::String *value);
      __property void set_Scheme(System::String *value);
      void addActionOfType(ObAMActionTypeMgd *action,
            ObAMActionMgd *value);
      void modifyActionOfType(ObAMActionTypeMgd *action,
            ObAMActionMgd *value);
      void removeActionOfType(ObAMActionTypeMgd *action,
            ObAMActionMgd *value);
};
```

Class ObAMAuthorizationRuleMgd

```
public __gc class ObAMAuthorizationMgd {
   public:
      ObAMAuthorizationRuleMgd();
      __property System::String *get_Name();
      __property bool get_Enabled();
      __property bool get_AllowTakesPrecedence();
      __property ObAMTimingConditionsMgd *get_TimingConditions();
```

```
        ObAMActionMgd *getActionOfType(ObAMActionTypeMgd *action,
             int index);
        __property ObAMAccessConditionsMgd
             *get_AllowAccessConditons();
        __property ObAMAccessConditionsMgd
             *get_DenyAccessConditons();
        __property System::String *get_AuthorizationScheme();
        __property int get_NumberOfSchemeParameters();
        ObAMParameterMgd *getSchemeParameter(int index);
        __property void set_IDFrom(ObAMAuthorizationRuleMgd *other);
        __property void set_Name(System::String *value);
        __property void set_Description(System::String *value);
        __property void set_Enabled(bool value);
        __property void set_AllowTakesPrecedence(bool value);
        __property void set_TimingConditions(
             ObAMTimingConditionsMgd *value);
        __property void set_AllowAccessConditions(
             ObAMAccessConditionsMgd *value);
        __property void set_DenyAccessConditions(
             ObAMAccessConditionsMgd *value);
        void addActionOfType(ObAMActionTypeMgd *action,
             ObAMActionMgd *value);
        void modifyActionOfType(ObAMActionTypeMgd *action,
             ObAMActionMgd *value);
        void removeActionOfType(ObAMActionTypeMgd *action,
             ObAMActionMgd *value)
};
```

Class ObAMAuthorizationExprMgd

```
public __gc class ObAMAuthorizatitonExprMgd {
   public:
        ObAMAuthorizationExprMgd();
        __property System::String *get_Expr();
        __property int get_DuplicateActionsPolicy();
        int getNumberOfActions(ObAMActionTypeMgd *type);
        ObAMActionMgd *getActionOfType(ObAMActionTypeMgd *type,
             int index);
        __property void set_Expr(System::String *value);
        void setDuplicateActionsPolicy(
             ObDuplicationActionPolicyMgd *value);
        void addActionOfType(ObAMActionTypeMgd *type,
             ObAMActionMgd *value);
        void modifyActionOfType(ObAMActionTypeMgd *type,
             ObAMActionMgd *value);
        void removeActionOfType(ObAMActionTypeMgd *type,
             ObAMActionMgd *value);
};
```

Class ObAMAuditRuleMgd

```
public __gc class ObAMAction_ValueTypeMgd {
   public:
      ObAMAuditRuleMgd();
      __property int get_NumberOfEvents();
      __property int get_NumberOfAttributes();
      ObAMAuditRule_EventTypeMgd *getEvent(int index);
      System::String *getAttribute(int index);
      __property void set_IDFrom(ObAMAuditRuleMgd *other);
      __property void set_AddEvent(
            ObAMAuditRule_EventTypeMgd *value);
      __property void set_AddAttribute(System::String *value);
      __property void set_RemoveEvent(
            ObAMAuditRule_EventTypeMgd *value);
      __property void set_RemoveAttribute(System::String *value);
};
```

Class ObAMAdminRuleMgd

```
public __gc class ObAMAdminRuleMgd {
   public:
      ObAMAdminRuleMgd();
      __property int get_NumberOfPersons();
      __property int get_NumberOfGroups();
      __property int get_NumberOfRoles();
      __property int get_NumberOfRules();
      ObAMIdentityMgd *getPerson(int index);
      ObAMIdentityMgd *getGroup(int index);
      System::String *getRole(int index);
      System::String *getRule(int index);
      __property void set_IDFrom(ObAMAdminRuleMgd *other);
      __property void set_AddRole(System::String *value);
      __property void set_AddPerson(ObAMIdentityMgd *person);
      __property void set_AddGroup(ObAMIdentityMgd *group);
      __property void set_AddRule(System::String *value);
      __property void set_RemoveRole(System::String *value);
      __property void set_RemovePerson(ObAMIdentityMgd *person);
      __property void set_RemoveGroup(ObAMIdentityMgd *group);
      __property void set_RemoveRule(System::String *value);
};
```

Class ObAMParameterMgd

```
public __gc class ObAMParameterMgd {
   public:
      ObAMParameterMgd();
      __property System::String *get_Value();
      __property System::String *get_Name();
      __property void set_Name(System::String *value);
      __property void set_Value(System::String *value);
};
```

Class ObAMPolicyMgd

```
public __gc class ObAMPolicyMgd {
   public:
      ObAMPolicyMgd();
      __property System::String *get_Name();
      __property System::String *get_Description();
      __property System::String *get_ResourceType();
      __property System::String *get_HostID();
      __property System::String *get_URLPattern();
      __property System::String *get_QueryString();
      __property int get_NumberOfOperations();
      __property int get_NumberOfResources();
      __property int get_NumberOfParameters();
      System::String *getOperation(int index);
      ObAMResourceMgd *getResource(int index);
      ObAMParameterMgd *getParameter(int index);
      __property ObAMAuthorizationExprMgd *get_AuthorizationExpr();
      __property ObAMAuthenticationRuleMgd
           *get_AuthenticationRule();
      __property ObAMAuditRuleMgd *get_AuditRule();
      __property void set_IDFrom(ObAMPolicyMgd *other);
      __property void set_Name(System::String *value);
      __property void set_Description(System::String *value);
      __property void set_ResourceType(System::String *value);
      __property void set_HostID(System::String *value);
      __property void set_URLPattern(System::String *value);
      __property void set_QueryString(System::String *value);
      __property void set_AuthenticationRule(
           ObAMAuthenticationRuleMgd *rule);
      __property void set_AuthorizationExpr(
           ObAMAuthorizationExprMgd *expr);
      __property void set_AuditRule(ObAMAuditRuleMgd *rule);
      __property void set_AddOperation(System::String *value);
      __property void set_AddResource(ObAMResourceMgd *resource);
      __property void set_AddParamter(ObAMParameterMgd *parameter);
      __property void set_ModifyResource(ObAMResourceMgd
           *resource);
      __property void set_RemoveOperation(System::String *value);
      __property void set_RemoveResource(ObAMResourceMgd
           *resource);
      __property void set_RemoveParameter(
           ObAMParameterMgd *parameter);
   };
```

Class ObAMPolicyDomainMgd

```
public __gc class ObAMPolicyDomainMgd {
   public:
      ObAMPolicyDomainMgd();
      __property System::String *get_Name();
      __property System::String *get_Description();
```

```
        __property bool get_Enabled();
        __property int get_NumberOfResources();
        __property int get_NumberOfAuthorizationRules();
        __property int get_NumberOfPolicies();
        ObAMResourceMgd *getResource(int index);
        ObAMAuthorizationRuleMgd *getAuthorizationRule(int index);
        __property ObAMAuthorizationExprMgd
             *get_DefaultAuthorizationExpr();
        ObAMPolicyMgd *getPolicy(int index);
        __property ObAMAuthenticationRuleMgd
             *get_DefaultAuthenticationRule();
        __property ObAMAuditRuleMgd *get_DefaultAuditRule();
        __property ObAMAdminRuleMgd *get_DelegateAdminRule();
        __property ObAMAdminRuleMgd *get_GrantAdminRule();
        __property ObAMAdminRuleMgd *get_BasicAdminRule();
        __property void set_IDFrom(ObAMPolicyDomainMgd *other);
        __property void set_Name(System::String *value);
        __property void set_Description(System::String *value);
        __property void set_Enabled(bool value);
        __property void set_DefaultAuthenticationRule(
             ObAMAuthenticationRuleMgd *value);
        __property void set_DefaultAuthorizationExpr(
             ObAMAuthorizationExprMgd *expr);
        __property void set_DefaultAuditRule(
             ObAMAuditRuleMgd *value);
        __property void set_DelegateAdminRule(
             ObAMAdminRuleMgd *value);
        __property void set_GrantAdminRule(ObAMAdminRuleMgd *value);
        __property void set_BasicAdminRule(ObAMAdminRuleMgd *value);
        __property void set_AddResource(ObAMResourceMgd *value);
        __property void set_AddAuthorizationRule(
             ObAMAuthorizationRuleMgd *value);
        __property void set_AddPolicy(ObAMPolicyMgd *value);
        __property void set_ModifyResource(ObAMResourceMgd *value);
        __property void set_ModifyPolicy(ObAMPolicyMgd *value);
        __property void set_ModifyAuthorizationRule(
             ObAMAuthorizationRuleMgd *value);
        __property void set_RemoveResource(ObAMResourceMgd *value);
        __property void set_RemoveAuthorizationRule(
             ObAMAuthorizationRuleMgd *value);
        __property void set_RemovePolicy(ObAMPolicyMgd *value);
};
```

Class ObAMAccessTestMgd

```
public __gc class ObAMAccessTestMgd {
   public:
      ObAMAccessTestMgd();
      __property System::String *get_URL();
      __property System::String *get_ResourceType();
      __property System::String *get_IPaddress();
      __property ObAMDateMgd *get_Date();
```

```
      __property ObAMTimeMgd *get_Time();
      __property int get_NumberOfOperations();
      __property int get_NumberOfUsers();
      System::String *getOperation(int index);
      ObAMIdentityMgd *getUser(int index);
      __property bool get_ShowDeterminingRules();
      __property bool get_ShowAllowed();
      __property bool get_ShowDenied();
      __property bool get_ShowMatchingPolicy();
      __property bool get_ShowMatchingExpr();
      __property void set_URL(System::String *value);
      __property void set_ShowDeterminingRules(bool value);
      __property void set_ResourceType(System::String *value);
      __property void set_AddOperation(System::String *value);
      __property void set_IPaddress(System::String *value);
      __property void set_Date(ObAMDateMgd *date);
      __property void set_Time(ObAMTimeMgd *time);
      __property void set_AddUser(ObAMIdentityMgd *value);
      __property void set_ShowAllowed(bool value);
      __property void set_ShowDenied(bool value);
      __property void set_ShowMatchingPolicy(bool value);
      __property void set_ShowMatchingExpr(bool value);
};
```

Class ObAMAccessTestResultsMgd

```
public _gc class ObAMAccessTestResultMgd {
   public:
      ObAMAccessTestResultsMgd();
      __property System::String *get_PolicyDomain();
      __property int get_NumberOfResults();
      ObAMAccessTestResultMgd *getResult(int index);
};
```

Class ObAMAccessTestResultMgd

```
public _gc class ObAMAccessTestResultMgd {
   public:
      ObAMAccessTestResultMgd();
      __property ObAMIdentityMgd *get_User();
      __property System::String *get_Policy();
      __property System::String *get_Expr();
      __property int get_NumberOfDeterminingRules();
      __property ObAMAccessTestResult_AuthzCodeMgd
           *get_AuthorizationStatus();
      System::String *getDeterminingRule(int index);
      __property bool get_Authoriszed();
};
```

Access System Configuration Objects

Java

Class ObAMHostIdentifier

```
public class ObAMHostIdentifier extends ObListElement {
   public String getName();
   public String getDescription();
   public int getNumberOfHostnames();
   public String getHostname(int index);
}
```

Class ObAMResourceType

```
public class ObAMResourceType {
   public String getName();
   public String getDisplayName();
   public boolean getCaseSensitiveMatching();
   public int getNumberOfOperations();
   public String getOperation(int index);
}
```

Class ObAMAuthenticationScheme

```
public class
ObAMAuthenticationScheme extends ObListElement {
   public static final int UNDEFINED = 0;
   public static final int NONE      = 1;
   public static final int BASIC     = 2;
   public static final int X509      = 3;
   public static final int FORM      = 4;
   public static final int EXT       = 5;
   public String getName();
   public String getDescription();
   public int getLevel();
   public int getChallengeMethod();
   public boolean getSSLrequired();
   public String getChallengeRedirectURL();
   public int getNumberOfChallengeParameters();
   public int getNumberOfPlugins();
   public String getChallengeParameter(int index);
```

```
      public ObAMAuthenticationPlugin getPlugin(
            int index);
      public boolean getEnabled();
      }
```

Class ObAMAuthenticationPlugin

```
   public class ObAMAuthenticationPlugin {
      public int getOrder();
      public String getName();
      public String getParameters();
   }
```

Class ObAMAuthorizationScheme

```
   public class ObAMAuthorizationScheme extends
   ObListElement {
      public String getName();
      public String getDescription();
      public String getLibrary();
      public int getNumberofUserParameters();
      public int getNumberofRequiredParameters();
      public int getNumberofOptionalParameters();
      public String getUserParameter(int index);
      public ObAMParameter getRequiredParameter(
            int index);
      public ObAMParameter getOptionalParameter(
            int index);
   }
```

Class ObAMMasterAuditRule

```
public class ObAMMasterAuditRule extends
   ObAMAuditRule {
      public static final int UNDEFINED = 0;
      public static final int INTEGER   = 1;
      public static final int MMDDYYYY  = 2;
      public static final int DDMMYYYY  = 3;
      public static final int ISO8601   = 4;
      public static final int YYYYMMDD  = 5;
      public static final int YYYYDDMM  = 6;
      public String getEventMapping(int eventType);
      public int getDateFormat();
      public char getEscapeCharacter();
      public String getRecordFormat();
}
```

C

Class ObAMHostIdentifier

```
typedef const void * ObAMHostIdentifier_t;
   const char *ObAMHostIdentifier_getName(
        ObAMHostIdentifier_t hostID);
   const char *ObAMHostIdentifier_getDescription(
        ObAMHostIdentifier_t hostID);
   int ObAMHostIdentifier_getNumberOfHostnames(
        ObAMHostIdentifier_t hostID);
   const char *ObAMHostIdentifier_getHostname(
        ObAMHostIdentifier_t hostID, int index);
```

Class ObAMResourceType

```
typedef const void * ObAMResourceType_t;
   const char *ObAMResourceType_getName(
        ObAMResourceType_t resType);
   const char *ObAMResourceType_getDisplayName(
        ObAMResourceType_t resType);
   int ObAMResourceType_getCaseSensitiveMatching(
        ObAMResourceType_t resType);
   int ObAMResourceType_getNumberOfOperations(
        ObAMResourceType_t resType);
   const char *ObAMResourceType_getOperation(
        ObAMResourceType_t resType, int index);

typedef const void * ObAMArrayOfResourceTypes_t;
   int ObAMArrayOfResourceTypes_numberOf(
        ObAMArrayOfResourceTypes_t array);
   ObAMResourceType_t ObAMArrayOfResourceTypes_get(
   ObAMArrayOfResourceTypes_t array,
        int index);
   void ObAMArrayOfResourceTypes_delete(
        ObAMArrayOfResourceTypes_t *pArray);
```

Class ObAMAuthenticationScheme

```
enum ObAMAuthenticationScheme_ChallengeMethod {
   ObAMAuthenticationScheme_UNDEFINED,
   ObAMAuthenticationScheme_NONE,
   ObAMAuthenticationScheme_BASIC,
   ObAMAuthenticationScheme_X509,
   ObAMAuthenticationScheme_FORM,
   ObAMAuthenticationScheme_EXT};

typedef const void * ObAMAuthenticationScheme_t;
   const char *ObAMAuthenticationScheme_getName(
      ObAMAuthenticationScheme_t scheme);
```

```
   const char *ObAMAuthenticationScheme_getDescription(
      ObAMAuthenticationScheme_t scheme);
   int ObAMAuthenticationScheme_getLevel(
      ObAMAuthenticationScheme_t scheme);
   ObAMAuthenticationScheme_ChallengeMethod
   ObAMAuthenticationScheme_getChallengeMethod(
      ObAMAuthenticationScheme_t scheme);
   int ObAMAuthenticationScheme_getSSLrequired(
      ObAMAuthenticationScheme_t scheme);
   const char *ObAMAuthenticationScheme_getChallengeRedirectURL(
      ObAMAuthenticationScheme_t scheme);
   int ObAMAuthenticationScheme_getNumberOfChallengeParameters(
      ObAMAuthenticationScheme_t scheme);
   int ObAMAuthenticationScheme_getNumberOfPlugins(
      ObAMAuthenticationScheme_t scheme);
   const char *ObAMAuthenticationScheme_getChallengeParameter(
      ObAMAuthenticationScheme_t scheme,int index);
   ObAMAuthenticationPlugin_tObAMAuthenticationScheme_getPlugin(
      ObAMAuthenticationScheme_t scheme, int index);

typedef const void * ObAMArrayOfAuthenticationSchemes_t;
   int ObAMArrayOfAuthenticationSchemes_numberOf(
        ObAMArrayOfAuthenticationSchemes_t array);
   ObAMAuthenticationScheme_t
        ObAMArrayOfAuthenticationSchemes_get(
        ObAMArrayOfAuthenticationSchemes_t array,int index);
   void ObAMArrayOfAuthenticationSchemes_delete(
        ObAMArrayOfAuthenticationSchemes_t *pArray);
   int ObAMAuthenticationScheme_getEnabled(
        ObAMAuthenticationScheme_t scheme);
```

Class ObAMAuthenticationPlugin

```
typedef const void * ObAMAuthenticationPlugin_t;
   int ObAMAuthenticationPlugin_getOrder(
        ObAMAuthenticationPlugin_t plugin);
   const char *ObAMAuthenticationPlugin_getName(
        ObAMAuthenticationPlugin_t plugin);
   const char *ObAMAuthenticationPlugin_getParameters(
        ObAMAuthenticationPlugin_t plugin);
```

Managed Code

Class ObAMHostIdentifierMgd

```
public __gc class ObAMHostIdentifierMgd {
```

```
   public:
      ObAMHostIdentifierMgd();
      __property System::String *get_Name();
      __property System::String *get_Description();
      __property int get_NumberOfHostnames();
      System::String *getHostname(int index);
};
```

Class ObAMResourceTypeMgd

```
public __gc class ObAMResourceTypeMgd {
   public:
      ObAMResourceTypeMgd();
      __property System::String *get_Name();
      __property System::String *get_DisplayName();
      __property bool get_CaseSensitiveMatching();
      __property int get_NumberOfOperations();
      System::String *getOperation(int index);
};
```

Class ObAMAuthenticationSchemeMgd

```
public __gc class ObAMAuthenticationSchemeMgd {
   public:
      ObAMAuthenticationSchemeMgd();
      __property System::String *get_Name();
      __property int get_Level();
      __property ObAMAuthenticationScheme_ChallengeMethodMgd
           *get_ChallengeMethod();
      __property bool get_SSLrequired();
      __property bool get_Enabled();
      __property System::String *get_ChallengeRedirectURL();
      __property int get_NumberOfChallengeParameters();
      __property int get_NumberOfPlugins();
      System::String *getChallengeParameter(int index);
      ObAMAuthenticationPluginMgd *getPlugin(int index);
   };
```

Class ObAMAuthenticationPluginMgd

```
public __gc class ObAMAAuthenticationPluginMgd {
   public:
      ObAMAuthenticationPluginMgd();
      __property int get_Order();
      __property System::String *get_Name();
      __property System::String *get_Parameters();
};
```

Class ObAMAuthorizationSchemeMgd

```
public __gc class ObAMAuthorizationSchemeMgd {
   public:
      ObAMAuthorizationSchemeMgd();
      __property System::String *get_Name();
      __property System::String *get_Description();
      __property System::String *get_Library();
      __property int get_NumberOfUserParameters();
      __property int get_NumberOfRequiredParameters();
      __property int get_NumberOfOptionalParameters();
      System::String *getUserParameter(int index);
      ObAMParameterMgd *getRequiredParameter(int index);
      ObAMParameterMgd *getOptionalParameter(int index);
};
```

Class ObAMMasterAuditRuleMgd

```
public __gc class ObAMMasterAuditRuleMgd {
   public:
      ObAMMasterAuditRuleMgd();
      System::String *getEventMapping(
           ObAMAuditRule_EventTypeMgd *eventType);
      __property ObAMMasterAuditRule_DateFormat get_DateFormat();
      __property const char get_EscapeCharacter();
      __property System::String *get_RecordFormat();
};
```

Class ObAMException

Topics below cover:

Java

```
   public class ObAMException extends
      com.oblix.access.ObAccessException{
      public static final int UNDEFINED           = 400;
      public static final int ADMIN_LOGIN_FAILED  = 401;
      public static final int NOT_AUTHORIZED      = 402;
      public static final int BAD_ARGUMENT        = 403;
      public static final int EXISTING_OBJECT     = 404;
```

```
        public static final int NO_OBJECT           = 405;
        public static final int BAD_MESSAGE         = 406;
        public static final int ALREADY_SET         = 407;
        public static final int FINALIZED           = 408;
        public static final int UNSUPPORTED_VERSION = 409;
        public static final int END_BEFORE_START    = 410;
        public static final int NO_SET_ADMIN        = 411;
        public ObAMException(int code);
        public ObAMException(int code, String p1);
        public ObAMException(int code, String p1, String p2);
        public ObAMException(int code, String p1, String p2,
             String p3);
        public ObAMException(int code, String p1, String p2,
             String p3, String p4);
        public ObAMException(int code, String p1, String p2,
             String p3, String p4, String p5);
        public int getCode();
        public String toString();
}
```

Class ObAccessException

C

```
enum ObAccessException_Code {
   . . .
      ObAccessException_AM_UNKNOWN   = 400,
      ObAccessException_AM_ADMIN_LOGIN_FAILED,
      ObAccessException_AM_NOT_AUTHORIZED,
      ObAccessException_AM_BAD_ARGUMENT,
      ObAccessException_AM_EXISTING_OBJECT,
      ObAccessException_AM_NO_OBJECT,
      ObAccessException_AM_BAD_MESSAGE,
      ObAccessException_AM_ALREADY_SET,
      ObAccessException_AM_FINALIZED,
      ObAccessException_AM_UNSUPPORTED_VERSION,
      ObAccessException_AM_END_BEFORE_START,
      ObAccessException_AM_UNSUPPORTED_OPERATION,
      ObAccessException_AM_NO_SET_ADMIN ,
      ObAccessException_AM_DATA_STORE_ERROR,
      ObAccessException_AM_READ_DATA_STORE_ERROR,
      ObAccessException_AM_INVALID_LDAP_FILTER,
      ObAccessException_AM_MISSING_REQUIRED_PARAM,
      ObAccessException_AM_INVALID_PARAM,
      ObAccessException_AM_NAME_REQUIRED,
      ObAccessException_AM_MODIFY_OBJECT_INVALID,
      ObAccessException_AM_INVALID_PROFILE_ATTRIBUTE,
      ObAccessException_AM_AUTHZ_SCHEME_CONFLICT,
```

```
      ObAccessException_AM_BAD_CHARACTER_DATA,
      ObAccessException_AM_CACHE_FLUSH_FAILED,
      ObAccessException_AM_AUTHN_SCHEME_PARAM,
      ObAccessException_AM_OBJECT_IN_USE,
      ObAccessException_AM_CANNOT_DELETE,
      ObAccessException_AM_POLICY_RESOURCE_TYPE_MISMATCH,
      ObAccessException_AM_INTERNAL_ERROR,
      ObAccessException_AM_INVALID_USER,
      ObAccessException_AM_INVALID_GROUP,
      ObAccessException_AM_FEATURE_NOT_SUPPORTED,
      ObAccessException_AM_INVALID_FAILURE_ACTION_ATTIBUTE,
      ObAccessException_MISSING_AUTHN_STEP
      ObAccessException_INVALID_AUTHZ_EXPR_SYNTAX,
      ObAccessExcpetion_AUTHZ_RULE_NOT_FOUND,
      ObAccessException_AUTHN_SCHEME_DISABLED,
      ObAccessException_INVALID_ACTION_TYPE,
      ObAccessException_INVALID_DUPLICATE_ACTIONS_POLICY
   };

typedef void (
        *ObAccessExceptionHandler2_t) (ObAccessException_t e);
   void ObAccessException_setHandler2(
        ObAccessExceptionHandler2_t handler);
   ObAccessExceptionCode_t ObAccessException_getCode(
        ObAccessException_t e);
   const char *ObAccessException_getParameter(
        ObAccessException_t e,int which);
   const char *ObAccessException_toString(
        ObAccessException_t e);
```

Class ObAccessExceptionMgd

For the enumerated list of exception codes, see “Class ObAccessException” on page 664.

Managed Code

```
public _gc class ObAccessExceptionMgd {
   public:
      ObAccessExceptionMgd();
      ObAccessExceptionMgd(ObAccessException *ex);
      __property ObAccessExceptionCode_t get_Code();
      System::String *getParameter(int index);
      System::String *getParameter(int index);
      __property System::String *get_String();
};
```

D SOAP and HTTP Client

Several NetPoint components, such as IdentityXML and AccessXML, allow you to gain access to NetPoint by using *SOAP* (Simple Object Access Protocol). To do this, you build a properly formatted SOAP request, with the NetPoint-related information contained within it.

SOAP provides a way to exchange information in a decentralized, distributed environment. It uses XML as a basis for its protocol, which consists of three parts:

- An envelope

  This defines a framework for describing what is in a message and how to process it. IdentityXML relies heavily upon this part.

- A set of encoding rules

  This provides a way to create application-defined data types. Both IdentityXML and AccessXML use this.

- A convention for communication

  SOAP defines a set of remote procedure calls and responses. Content for these can be established using the encoding rules. SOAP could be used in combination with almost any protocols. For NetPoint, the focus is on its use in combination with HTTP and servers.

A full discussion of the protocol can be found at:

http://www.w3.org/TR/SOAP/

Though SOAP provides the means to communicate with NetPoint, it is still necessary to transport the message content using the Web to the COREid or Access System Server that will process it. This requires the use of an *HTTPClient.* The HTTPClient is an application that simulates the HTTP communication capabilities of a browser, without an HTML presentation.

Though such a client could be written from scratch, toolkits are available that provide the necessary components. One such toolkit is available from Innovation:

http://www.innovation.ch/java/HTTPClient/

The toolkit is free and internally documented. It includes support for the request methods HEAD, GET, POST and PUT, and contains modules that support automatic handling of authorization, redirection requests, and cookies.

You use the toolkit to provide the HTTP communication modules that will be the back end of an HTTPClient that you write. The front end of your client will have these features:

- Host Identification

  You need to be able to identify the full Host URL that you want to communicate to, including the port number, and provide this information to the back end.

- Data Transmission

  You need a way to pick up and send data to the host. The data to be picked up could be the entire SOAP envelope with data, or just the data, with the envelope being applied by your client, or could be assembled almost entirely within the client. You provide the data to the back end for transmission, and expect the back end to return the response to you. You can include modules in the back end that will work with redirection responses and maintain cookies to support single sign-on.

- Response Interpretation

  You need a way to parse and use the information returned by NetPoint.

A sample of just such a client is provided at:

$COREid_install_dir/unsupported/integsvcs

in the file ObSoapClient.java. You need to compile this file into a class version, within the HTTPClient build environment.

The resulting example allows you to send a selected request file to a selected port of a selected host. The command line arguments are:

java ObSoapClient -h hostname -p port -f file

where *hostname* is the URL you want to communicate with, `port` is the port number, and *file* is the name of the request file you want to send. The response is displayed to the screen. You will probably want to pipe this to a file, or modify the example to print to a file you name on the command line.

The file ObSoapClient.pl provides a similar example for use with PERL.

Several example request files are provided for you, also in the location:

$COREid_install_dir/unsupported/integsvcs

You will not be able to use these files as is. You will need to change at least the login and password information in each one to information matching a valid user on the system you are trying to access. And, you will probably need to change the uid information in each file to match your directory structure and content.

E Managed Helper Classes

This appendix contains information on managed helper classes for the Access Server and Access Management APIs.

Managed Helper Classes for the APIs

```
Namespace: Oblix.Access.Common


      /*
      * ObDictionary class methods allow the application to
      *
      *    - provide .NET style dictionary access to Java style ObMap object
      */
      public __gc class ObDictionary :
         public System::Collections::IDictionary,
         public System::IDisposable,
         public System::ICloneable
      {
      private:
         ObMap __nogc* _map;

      public:
          ObDictionary();
         ObDictionary(const ObMap& map);
          // This constructor takes over the memory of the map object
         ObDictionary(ObMap *map);
         ~ObDictionary();

         //ICloneable
         Object* Clone();

         //IDisposible
         void Dispose();
         void Dispose(bool disposing);

          //IDictionary
         virtual void CopyTo(System::Array* ar, int count);
          virtual IEnumerator* IEnumerable::GetEnumerator() {return
GetEnumeratorImpl();}
```

```
        virtual IDictionaryEnumerator* IDictionary::GetEnumerator() {return
GetEnumeratorImpl();}
        virtual IDictionaryEnumerator *GetEnumeratorImpl();

        bool Contains(System::Object* key);
        __property bool get_IsFixedSize();
        __property bool get_IsReadOnly();
        __property bool get_IsSynchronized();
        __property int get_Count();
        __property System::Collections::ICollection* get_Keys();
        __property System::Collections::ICollection* get_Values();
         __property ObMap *get_Map();

         // This class is not thread safe, as a result there is no syncroot
object. This method
         // returns a NULL object.
        __property System::Object* get_SyncRoot();

        virtual void Add(System::Object* key, System::Object* value);
        virtual void Clear(void);
        virtual void Remove(System::Object* key);

        //__property virtual void set_Item(System::String* key, System::String*
value);
        __property virtual void set_Item(System::Object* key, System::Object*
value);
        __property virtual System::Object* get_Item(System::Object* key);
        //__property virtual System::String* get_Item(System::String* key);

        //Type specific overloads
        void Add(System::String* key, System::String* value);
        bool Contains(System::String* key);
     };

     __gc class ObDictionaryEnumerator : public IDictionaryEnumerator
     {
     private:
        int _iCurrent;
         int _entrySize;
        DictionaryEntry _ar[];
     public:
        ObDictionaryEnumerator(ObDictionary* dict);

        //IDictionaryEnumerator
        Object* get_Current();
        DictionaryEntry get_Entry();
        Object* get_Key();
        Object* get_Value();
        bool MoveNext();
        void Reset();
     };

/*
```

```
 * ObConfigMgd functions allow the application to
 *
 *    - initialize the Access API from a configuration file,
 *    - shutdown: delete resources used by the API,
 *    - get information from the Access API configuration, including
 *      - sessionTimeout: the maximum lifetime in seconds for a user session
 *      - idleTimeout:    the maximum period in seconds allowed between
authorization events
 *    - map user session error numbers to messages
 *    - get current version number for the Access Server SDK
 *    - get current version of NetPoint Access Protocol or NAP version.
 */
public __gc class ObConfigMgd {

        public:
            static void initialize(System::String *configDir);
            static void initialize();
            static void shutdown();
            __property static ObDictionary *get_AllItems();
            __property static int get_NumberOfItems();
            static System::String *getItem(System::String *name);
            static System::String *getErrorMessage(int err);
            __property static System::String *get_SDKVersion();
            __property static System::String *get_NAPVersion();

        private:

        };

/*
 * Access Exception Implementation Objects
 * An ObAccessExceptionImpl object is thrown when a problem is detected by the
Access API
 * implementation methods. Access codes are defined in obaccess_api_defs.h. The
mapAAAStatus()
 * class method maps an ObAAAStatus returned by an ObAAAServiceClient method into
a exception
 * code.
 */
public __gc class ObAccessExceptionMgd : public System::Exception {
        public:
            ObAccessExceptionMgd();
            ObAccessExceptionMgd(ObAccessException *ex);

           //Cleanup
            ~ObAccessExceptionMgd();

           //IDispose()
            void Dispose();
           void Dispose(bool disposing);

            // getters and setters
            __property ObAccessExceptionCode_t get_Code();
```

```
            System::String *getParameter(int index);
            System::String *getCodeString(ObAccessExceptionCode_t code);
            __property System::String *get_String();

        private:

            /// Unmanaged Oblix object.
            ObAccessException __nogc *_exception;
         };
    }
}
```

Index

Symbols

A

C

E

F

G

H

I

J

N

O

P

R

S

T

U

V

W

X