Access Control Programmer's Guide

Netscape Enterprise Server/FastTrack Server

Version 3.0/3.01

Netscape Communications Corporation ("Netscape") and its licensors retain all ownership rights to the software programs offered by Netscape (referred to herein as "Netscape Software") and related documentation. Use of the Netscape Software is governed by the license agreement accompanying such Netscape Software. The Netscape Software source code is a confidential trade secret of Netscape and you may not attempt to decipher or decompile Netscape Software or knowingly allow others to do so. Information necessary to achieve the interoperability of the Netscape Software with other programs may be obtained from Netscape upon request. Netscape Software and its documentation may not be sublicensed and may not be transferred without the prior written consent of Netscape.

Your right to copy Netscape Software and this documentation is limited by copyright law. Making unauthorized copies, adaptations, or compilation works (except for archival purposes or as an essential step in the utilization of the program in conjunction with certain equipment) is prohibited and constitutes a punishable violation of the law.

THIS DOCUMENTATION IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND. IN NO EVENT SHALL NETSCAPE BE LIABLE FOR ANY LOSS OF PROFITS, LOSS OF BUSINESS, LOSS OF USE OR DATA, INTERRUPTION OF BUSINESS, OR FOR INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES OF ANY KIND, ARISING FROM ANY ERROR IN THIS DOCUMENTATION.

Netscape may revise this documentation from time to time without notice.

Copyright © 1997 Netscape Communications Corporation. All rights reserved.

Netscape Communications, the Netscape Communications logo, Netscape, and Netscape News Server are trademarks of Netscape Communications Corporation. The Netscape Software includes software developed by Rich Salz, and security software from RSA Data Security, Inc. Copyright © 1994, 1995 RSA Data Security, Inc. All rights reserved. Other product or brand names are trademarks or registered trademarks of their respective companies.

Any provision of Netscape Software to the U.S. Government is with "Restricted rights" as follows: Use, duplication or disclosure by the Government is subject to restrictions set forth in subparagraphs (a) through (d) of the Commercial Computer Restricted Rights clause at FAR 52.227-19 when applicable, or in subparagraph (c) (1) (ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013, and in similar clauses in the NASA FAR Supplement. Contractor/manufacturer is Netscape Communications Corporation, 501 East Middlefield Road, Mountain View, California 94043.

You may not export the Software except in compliance with applicable export controls. In particular, if the Software is identified as not for export, then you may not export the Software outside the United States except in very limited circumstances. See the end user license agreement accompanying the Software for more details.


Recycled and Recyclable Paper

The Team:
Engineering:Chris Apple, Mike Barbarino, Mike Belshe, Jim Black, Fred Cox, George Dong, Alex Feygin, Alan Freier, Andy Hakim, Warren Harris, John K. Ho, Ari Luotonen, Mike McCool, Rob McCool, Chuck Neerdaels, Howard Palmer, Ben Polk, Aruna Victor
Marketing: Mike Blakely, Atri Chatterjee, Ben Horowitz, David Pann
Publications: Guy K. Haas
Quality Assurance: Saleem Baber, Roopa Cheluvaiah, Shvetal Desai, Noriko Hosoi, Teresa Hsiao, Pramod Khincha, Joy Lenz, Rajesh Menon, Jun Tong, Cathleen Wang, Carol Widra, Ayyaz Yousaf
Technical Support: John Benninghoff, Brian Kendig, Anthony Lee-Masis, Trevor Placker, Bill Reviea, Dan Yang

Netscape Enterprise Server/FastTrack Server Version 3.0/3.01

©Netscape Communications Corporation 1997
All Rights Reserved

Printed in USA
97 96 10 9 8 7 6 5 4 3 2 1

Netscape Communications Corporation 501 East Middlefield Road, Mountain View, CA 94043

Contents

About This Guide

The *Access Control Programmer's Guide* documents the API libraries for managing access to Netscape web servers. (For an explanation of access control and ACLs, see Chapter 1, "Understanding Access Control".)

Who Should Read This Guide?

This guide is for programmers who want to manage and extend the access control features in Netscape web servers through server plug-ins.

What's in This Guide?

This guide explains how to use the Access Control API in Netscape web servers to manage access control programmatically. The Access Control API consists of data structures and functions used to manage access control.

Table 1 describes each chapter in more detail.

Table 1 Finding information in this manual

To do this:	See this chapter:
Learn more about access control and Netscape web servers	Chapter 1, "Understanding Access Control"
Understand the Access Control API libraries and how to write your own plug-ins	Chapter 2, "Using the Access Control API"

Table 1 Finding information in this manual (Continued)

To do this:	See this chapter:
Understand the standard attributes and how servers get these attributes	Chapter 4, "How the Standard Attributes Work"
Define your own attributes for use during authentication	Chapter 5, "Defining Attributes for Authentication"
Define your own authentication methods or types of authentication databases	Chapter 6, "Defining Methods and Database Types"
See examples of using custom attributes	Chapter 7, "Examples of LAS Modules"
Look up the description of a data structure	Chapter 8, "Data Types and Structures"
Look up the description of a function	Chapter 9, "Functions and Function Typedefs"

Conventions in This Book

The following table lists typographical conventions used in this book.

Table 0.1 Typographical Conventions

Convention	Description
`Monospaced font`	This typeface is used for sample code and code listings, API and language elements (such as function names and class names), filenames, pathnames, directory names, HTML tags, and any text that must be typed on the screen. (Monospaced italic font is used for placeholders embedded in code.)
Italics	Italics type is used for book titles, emphasis, variables and placeholders, and words used in the literal sense.

Table 0.1 Typographical Conventions

Convention	Description
Boldface	Boldface type is used to introduce new terms.
|	The vertical bar is used as a separator for user interface elements. For example, File | New means you should choose New from the File menu; Server Status | Log Preferences means you should click the Server Status button in the Server Manager and then click the Log Preferences link.

Chapter

1

Understanding Access Control

Before you use the Access Control API for Netscape web servers, you should understand some basic definitions and concepts in access control. If you are already familiar with this material, you can skip ahead to Chapter 2, "Using the Access Control API".

What Is Access Control?

Access control is defined in terms of **resources** and **subjects**:

- A server **resource** is the entity that is under access control.
- A **subject** is the entity that initiates access to a server resource.

For example, suppose a user named Mozilla accesses the home page for a web server. Mozilla is the subject and the home page is the resource.

The subject can be described by an open-ended set of named **attributes** and their corresponding values. Attributes that describe a subject might include the subject's username, the groups in which the subject belongs, and the IP address or host name of the machine that the subject is sending the request from.

The values of these attributes help to identify the subject and can be used to determine what kind of access the subject is allowed. For example, access to a server resource might be restricted to the subjects with certain usernames or the host names where the request originated.

Authentication is the process of acquiring and verifying the values of the specified attributes of the subject. For example, authentication might involve prompting the user to login with a username and password, and then looking in a database to verify that the user's password is correct.

Authorization is the process of checking the rights (or permissions) to the server resource that are allowed for the subject. For example, a subject might be allowed read access but not write access to a server resource.

An **access control entry (ACE)** specifies the rules for accessing a given server resource. There are two types of access control entries:

- **Authentication ACEs** define how the subject is identified. Typically, authentication ACEs specify the following:
    - The attributes that need to be obtained from the subject—for example, the subject's username or IP address
    - The method(s) for obtaining these attributes—for example, by using basic Hypertext Transfer Protocol (HTTP) authentication or by using Secure Sockets Layer (SSL) authentication
    - The authentication database, if needed—for example, the database containing user information
- **Authorization ACEs** define the rights allowed or denied for a particular subject or subjects. For example, an authorization ACE might specify that all users except the user named Mozilla have write access to the home page.

An **access control list (ACL)** is an ordered list of ACEs. An ACL can contain both types of ACEs.

In Netscape web servers, you can create, modify, and remove access control lists for different resources through the administration server interface. These ACLs are stored in the `generated.https-`*`server_id`*`.acl` file under the *`server_root`*`/httpacl` directory. This file is generated whenever access control settings are changed.

The following is an example of an ACL file:

```
# File automatically written
#
# You may edit this file by hand
#

version 3.0;

acl "path=/usr/netscape/suitespot/docs/index.html";

authenticate (user,group) {
   method="basic";
   database="employees";
};

allow (all)
   user = "mozilla";

deny (write, delete)
   user = "all";
```

The ACL in this file controls access to the resource `/usr/netscape/suitespot/docs/index.html`. The ACL consists of three ACEs:

- The first ACE is an authentication ACE. It specifies that the server needs to get the `user` and `group` attributes of the subject attempting to access the resource. The `basic` method is used to get the values of these attributes. The `employees` database is used to find information on the subject.
- The second and third ACEs are authorization ACEs. The second ACE specifies that the subject with the username `mozilla` has all access rights to the resource. The third ACE specifies that all other users are denied the ability to write to and delete the specified resource.

For more information on the syntax used in ACL files, see your server administrator's guide (provided with your web server) or Appendix A, "ACL Files."

How Access Control Works in the Server

In Netscape web servers, you can associate ACLs with resources in the following ways:

- In an ACL file, specifying an ACL with the name `path=<path_to_your_resource>`.

- In an ACL file, specifying an ACL with the name `uri=<uri_of_your_resource>`.
- In the `obj.conf` file, specifying the `check-acl` built-in function in a `PathCheck` directive.

When processing an incoming request, Netscape web servers follow these general steps to determine if the requestor has the authorization to access the resources:

1. The incoming Uniform Resource Identifier (URI) is parsed and name translation is performed (for example, name translation directives are applied at this step).

   At this stage, the server checks the execute and search permissions on each container object (such as a directory) that needs to be traversed to access the requested resource.

   As the server traverses the path, it collects the ACL of each object in the path to the resource.

2. After traversing the path to the resource, the server checks the object type of the resource. Here, the server checks the ACLs to determine which attributes (if any) restrict access to the resource. For example, the resource's access rights might be restricted by a user name and password or by the IP address of the subject requesting access to the resource.

3. Next, the server calls the appropriate evaluation function. At this point, the server begins to test the access control:

   - First, if authentication is required, the server gets authentication information. The data sources containing authentication information are invoked. The server uses an attribute getter function to get the value of an attribute from the subject (for example, the requestor's login name and password or certificate). The getter function also retrieves the actual value of the attribute from the database specified in the getter function.

   - Next, the information supplied by the requestor is validated by the attribute getter function. The server compares the information given by the subject to entries in the database. If the subject's information matches entries in the database, the subject's identity is authenticated and the function returns the information for evaluation.

- Finally, the authorization statements in the collected ACLs are evaluated. The server checks the subject's authenticated information against the ACEs of the resource to determine what rights (such as read or write) are available to the subject.

How Access Control Processes a Request

Granting access rights to a resource starts with a subject's request to access a resource on the server. For example, an incoming URI comes from a subject named Mozilla in the form of a request for access to `/usr/netscape/suitespot/docs/index.html`.

This URI is then parsed and Resource Managers for the container objects at each level of the pathname are called to acquire any applicable container ACLs. These container objects are tested for execute/search access inline to the terminal object. When the terminal object (`index.html`) is reached, the ACLs are accumulated for overall evaluation. This means that ACLs are collected and tested for each folder that Mozilla must traverse to get to index.html.

Once the terminal object is reached, the server calls the appropriate evaluation function depending on how the ACL is defined.

Suppose the ACL is defined as:

```
acl "path=/export/netscape/suitespot/docs/index.html";
authenticate (user, group) {
      method="basic";
      database="ldap";
};
allow (read)
      user=all
deny (read)
      user="Mozilla";
```

The server determines that the `user` attribute must be validated before Mozilla is authorized to access to the resource. The server then calls the evaluation function for the user attribute.

The evaluation function checks the authorization ACE in the ACL to see if the attribute pattern for granting access is `anyone`. Because "`anyone`" means any unauthenticated user, this step precedes authentication and the calling of the attribute getter function. (If the ACL expression is user="`anyone`", then the user

is granted access at this point.) The ACL statement does not include the attribute pattern `anyone`, so the evaluation function advances to the authentication process.

To authenticate the `user` value, the evaluation function calls the attribute getter function for the `user` attribute. Because the authentication statement for the user attribute in the ACL has `method=basic` and `database=ldap`, the attribute getter that will be called is the `user` attribute getter function that is registered under `method=basic` and `database=ldap`. The attribute getter function is typically dependent on method/database type and can be defined for any method or any database type.

The attribute getter function for the `user` attribute retrieves the username and password from Mozilla's request and uses them to authenticate to the LDAP database. If a match is found, the function makes the username available to the evaluation function. (If no match is found, Mozilla is denied access to the resource at this point.)

The evaluation function now compares the authenticated `user` value against the authorization ACE in the ACL for `/export/netscape/suitespot/docs/index.html` to determine Mozilla's access rights to the resource.

The authorization ACE first states that all authenticated users (`user="all"`) have access rights to read `/export/netscape/suitespot/docs/index.html`. It then stipulates that subjects with the `user` value `Mozilla` are denied read access to the resource. The evaluation function takes the returned `user` value and compares it against the authorization ACE and finds that Mozilla is denied read access. Mozilla is denied access to `/export/netscape/suitespot/docs/index.html` at this point.

The evaluation function then clears the cache of any stored data used to validate Mozilla's request for `/export/netscape/suitespot/docs/index.html`

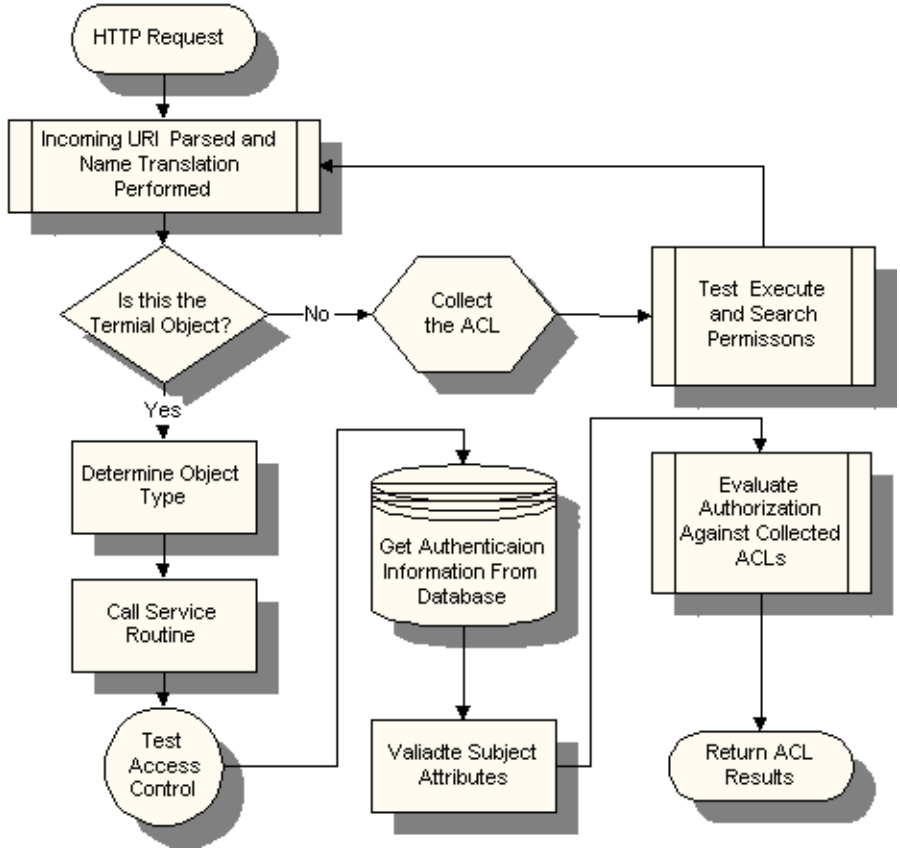


Figure 1.1 Processing a request

How Access Rights Work

In Netscape web servers, there are two types of access rights: generic rights and specific rights. Generic rights include:

- read
- write
- execute (search)
- create
- delete
- append

These rights can apply to any type of resource (in other words, they are not specific to web documents and directories on a web server).

Specific rights apply to specific types of resources. For example, the specific rights to a web document might include http_get or http_put. These rights do not need to be a limited set of predefined rights; a specific right can be any string value.

How the Loadable Authentication Service (LAS) Works

ACEs are evaluated by **Loadable Authentication Service (LAS)** modules. The body of an ACE uses the form:

```
<attribute><operator><pattern>
```

`<pattern>` is typically a constant. To evaluate this expression, the LAS needs to get the value of `<attribute>`. The following example of an ACE is part of an allow or deny statement:

```
user="mozilla";
```

An LAS module defines the following functions:

- An **evaluation function** for evaluating the value of the attribute against the ACE to determine if the subject should be allowed access to the resource.

- A **flush function** for freeing any cached attribute data when an ACL is freed from memory.

Getting and Evaluating Attribute Values

As part of the process of evaluating the attribute, the LAS module gets the value of the attribute from the subject.

The ACL can specify:

- an authentication method, which can be used to get the value of the attribute from the subject
- an authentication database, which can be used to get the actual value of the attribute.

To get the value of an attribute, an LAS module can call an **attribute getter** function. An attribute getter function retrieves the value of an attribute using a specified authentication method and (if necessary) compares it against a specified database.

For example, the `user` attribute might have two attribute getter functions: one associated with the `basic` authentication method, and one associated with the `ssl` authentication method.

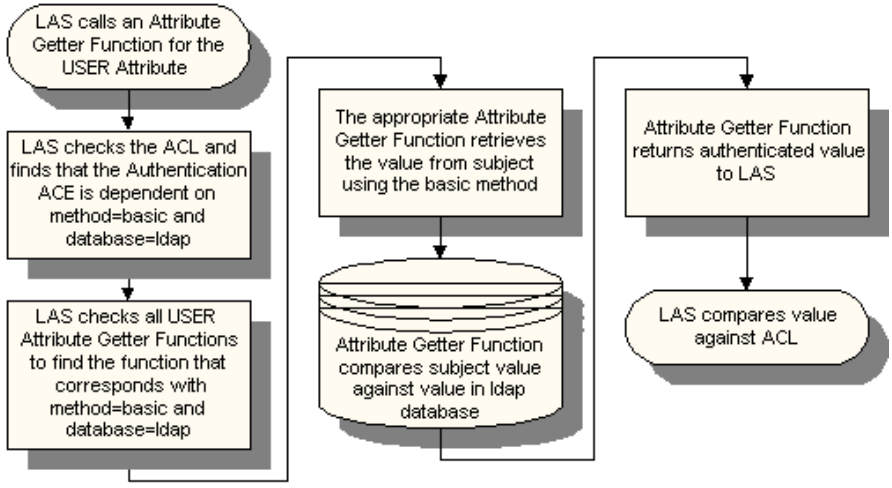


Figure 1.2 Relationship between Attribute, Method, and Database

Some attributes getter functions check the value of the attribute against the value in an authentication database. (For example, an attribute named ACL_ATTR_USER_ISMEMBER might check to see if a user is a member of a specified group.) This process is called **Authentication** and is a part of the attribute getter function. If the value of the attribute must be compared against data in a particular authentication database, the name of the database must be specified in the ACL.

The attribute getter function then returns the authenticated value to the evaluation function. The evaluation function compares the subject's value against the value in the ACLs, it then grants or denies access to the resource. This process is called **Authorization**.

After Authorization, the LAS module calls a flush function to clear the cache of any stored attribute data.

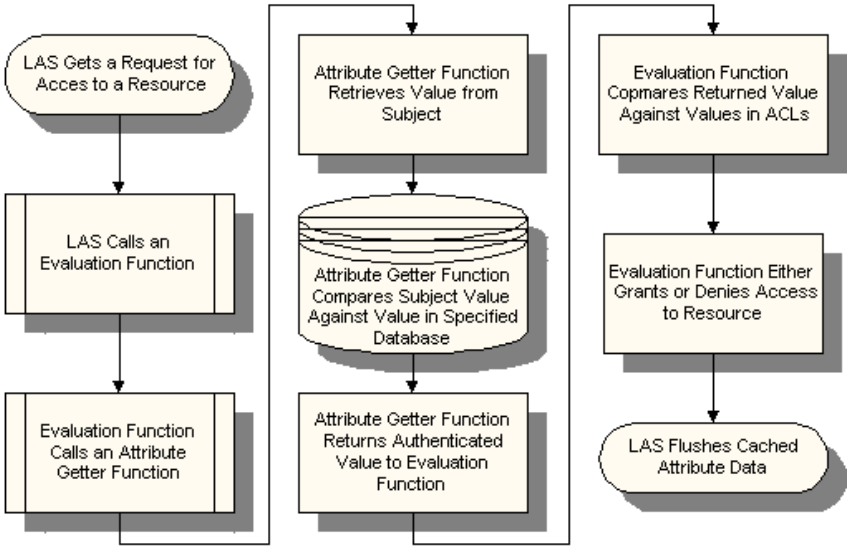


Figure 1.3 LAS Module Processing a Request

Standard LAS Modules

Netscape web servers include a set of standard LAS modules that are used for authentication. The attributes defined by these modules include:

- timeofday
- dayofweek
- ip
- dns
- group
- user

During startup, the server registers these LAS modules. Functions for evaluating these attributes and flushing any cached attribute data are also registered.

The server then registers the standard authentication methods and database types used. The server registers the following types of methods:

- Basic
- SSL

The server registers the following types of authentication databases:

- LDAP (Lightweight Directory Access Protocol)

Finally, the server registers the functions for getting the standard attributes. For details about these functions, see Chapter 4, "How the Standard Attributes Work".

Defining Your Own Attributes and LAS Modules

Using the Access Control API, you can define your own attributes to be used for authentication. You need to write your own LAS function if you want to define a new ACL attribute or you want to replace the behavior of an existing LAS function.

For example, If you want to evaluate the subject with a non-standard attribute (such as the subject's age or e-mail address) you can write a new attribute getter function to retrieve the new attribute. You would then need to write a

new evaluation function to call the new attribute getter function and evaluate the attribute against your database or modify an existing LAS to call your new function. For details, see Chapter 5, "Defining Attributes for Authentication."

You can also define your own authentication methods and databases. The standard database registered during startup is LDAP. If you want to use your own database type (such as ODBC or RDBMS) you must register it in the obj.conf file. In addition, if you wish to use another method besides basic HTTP or SSL to retrieve the attribute, you must also register it in the obj.conf file. For details, see Chapter 6, "Defining Methods and Database Types."

How Certificate-Based Authentication Works

Authentication is provided through digital authentication certificates, which are very difficult to falsify. Certificates are nontransferable, nonforgable files that act as a sort of digital identity badge or passport to help insure that users or computers are who they say they are. Certificates hold the following information:

- the holder's name, organization, address, and so on
- the holder's public key
- the certificate's validity dates
- the certificates serial number

You purchase or otherwise receive a certificate from a third party that both communicating parties already trust. This third party is known as a certificate authority, or CA.

A certificate authority can be a company that sells certificates over the Internet, or it can be a department responsible for issuing certificates for your company's intranet. You decide which certificate authorities you trust enough to serve as verifiers of other people's identities.

Both clients and servers have certificates. When a server sends its certificates to a client, the process is called **server authentication**. When a client sends a certificate to a server, the process is called **client** (or user) **authentication**.

Server Authentication

When SSL is enabled on a server, server authentication is accomplished with these steps:

1. A client requests a connection with an SSL-enabled server.
2. The server signs but does not encrypt its certificate and sends it to the client.
3. The client uses the server's public key, which is included in the certificate file, to verify that the owner of the certificate is the same as the one who signed it.
4. The client checks whether the certificate's issuer is one that it accepts. If so, the client proceeds to the next step; otherwise, the client's program informs its user that this certificate was issued by an unknown certificate authority.
5. The client compares the information in the certificate with the information it just received about the server. If all information matches, the client accepts the server as authenticated.

Client Authentication

Client authentication is accomplished with these steps:

1. A client (such as a browser) requests a connection with the server.
2. The server is authenticated or not (through the process of server authentication).
3. The client signs but does not encrypt its certificate and sends it to the server.
4. The server uses the client's public key, which is included in the certificate file, to verify that the owner of the certificate is the same one who signed it.
5. The server attempts to match the certificate authority to a trusted certificate authority. If the client's certificate is not listed as trusted, the server ends the transaction, logs an error, and returns this message to the client: "The server cannot verify your certificate."

6. If the client's certificate authority is trusted, the server moves to the next step in the authentication process. The server needs to match the information from the certificate with an entry in an LDAP directory to further identify and authenticate the user. If all information matches, the server accepts the client as authenticated. If entries in your database contain certificates rather than information, the server compares the sent certificate to the one in the database. If they match, the server grants the client access.

This manual provides more information about certificate mapping and certificate based authentication and authorization in Chapter 4.

Chapter

2

Using the Access Control API

The Netscape web servers include an API that you can use to manipulate access control and ACLs.

Access Control API

The Access Control API provides functions that allow you to extend the access control features in Netscape web servers.

With the Access Control API, you can manipulate ACLs, read and write ACL files, and evaluate and test access control to resources on the server. You can also define your own attributes for authentication and your own authentication methods and databases.

The rest of this chapter describes how to write plug-ins that use the Access Control API.

For details on defining your own attributes for authentication, see Chapter 5, "Defining Attributes for Authentication".

For information on defining your own authentication methods or databases, see Chapter 6, "Defining Methods and Database Types".

NSAPI, ACL API, and Access Control

NSAPI is a programming interface that allows you to customize the process of handling incoming HTTP requests. Access control is part of this process. If you want to use NSAPI to control access to resources, you can associate ACLs with resources by using the check-acl function in the PathCheck directive of the `obj.conf` file. For information on the check-acl function and the PathCheck directive, see the *NSAPI Programmer's Guide*.

Note If you are not already using NSAPI, Netscape recommends that you use Web Application Interface (WAI) technology or the Access Control (ACL) API instead. Use WAI for writing web applications and ACL for customizing user authentication.

Writing Your Own Plug-ins

To use the Access Control API, you can write standard server plug-ins. (See the *NSAPI Programmer's Guide* for more information on writing server plug-ins.) The header files for the Access Control API are in the following directory:

server_root`/include/nsacl (on UNIX)`

server_root`\include\nsacl (on Windows NT)`

This directory contains the following header files:

- `aclapi.h` is the main header file for the Access Control API. Make sure to include this file when writing your plug-ins:

  ```
  #include "aclapi.h"
  ```

  Because this header file includes the other header files in the directory, you do not need to explicitly include the other header files in your plug-in.

- `acldef.h` defines some of the data types and structures in the Access Control API.

- `nserrdef.h` defines the structures used for error reporting.

- `plistdef.h` declares functions used to create, manipulate, and destroy property lists. In the Access Control API, property lists are used to represent authentication information and information about the subject requesting access and the resource being accessed.

When compiling your plug-in, keep the following tips in mind:

- Add the following include directories to your makefile:
  - On UNIX, *server_root*/include
  - On Windows NT, *server_root*\include
- Define the following:
  - On Solaris, define XP_UNIX, SOLARIS, and _REENTRANT.
  - On IRIX, define NET_SSL and IRIX, and specify the following flags: -D_SGI_MP_SOURCE -fullwarn.
  - On HP-UX, define NET_SSL and HPUX, and specify the following flags: -D_HPUX_SOURCE -Aa +DA1.0 +z.
  - On Windows NT, define XP_WIN32 and WIN32.
- Add the following options to the linker:
  - On Solaris, specify the -G options to the ld command.
  - On IRIX, specify the -32 -shared options to the ld command.
  - On HP-UX, specify the -b option to the ld command.

Working with ACLs in Memory

The Access Control API allows you to read ACL files into memory and manipulate ACLs as in-memory structures.

The following table summarizes these structures and data types.

Table 2.1 Data types and structures representing ACLs

ACL component	Example	Corresponding data type/structure
Access control entry (ACE)	`allow (read)` `  user = "mozilla";`	`ACLExprHandle_t`

Access control list (ACL)	`acl "uri=/index.htm"`
	`...`
	`allow (read)`
	`  user = "mozilla";`
	`...`
		`ACLHandle_t`

ACL list	`acl "uri=/index.htm"`
	`...`
	`acl "uri=/docs"`
	`...`
		`ACLListHandle_t`

Parsing ACL Files

In order to manipulate ACLs in memory, you first need to parse the ACL file into an `ACLListHandle_t` structure, which represents an ACL list in memory. To parse an ACL file, call the `ACL_ParseFile()` function.

The following example converts the ACL file `/usr/netscape/suitespot/httpacl/generated.https-rtfm.acl` into an `ACLListHandle_t` structure.

```
#include "aclapi.h"

...

ACLListHandle_t *acl_list;
NSErr_t err;
char *filename =
   "/usr/netscape/suitespot/httpacl/generated.https-rtfm.acl";

...

/* Parse the ACL file into an ACLListHandle_t structure. */
acl_list = ACL_ParseFile(&err, filename);

/* Can now call functions to manipulate ACLs through acl_list. */
```

. . .

Manipulating ACL Lists

You can call functions in the Access Control API to do the following:

- **Get a specific ACL from the ACL list.**

  To get an ACL from an ACL list, call the `ACL_ListFind()` function. The function returns an `ACLHandle_t` structure, which represents a single ACL in memory.

- **Remove an ACL from the ACL list.**

  To remove an ACL from an ACL list, call the `ACL_ListAclDelete()` function. This only affects the ACL list in memory. To permanently remove an ACL from an ACL list in a file, call the `ACL_FileDeleteAcl()` function.

- **Append an ACL to the ACL list.**

  To add an ACL to an ACL list, call the `ACL_ListAppend()` function.

- **Concatenate different ACL lists together into a single ACL list.**

  To concatenate two different ACL lists together, call the `ACL_ListConcat()` function.

- **Get a list of the names of the ACLs in the ACL list.**

  To get the names of the ACLs in an ACL list, call the `ACL_ListGetNameList()` function.

Evaluating ACLs

During the course of processing an incoming request, you can determine if a user has certain rights to the requested resource by calling the `ACL_SetupEval()` function.

In addition to the name of the user and the ACL list used, you need to specify the rights that you want to check. As described in "How Access Rights Work", access to any type of resource is defined by a set of generic access rights. These rights include:

- `read`
- `write`
- `execute`
- `delete`
- `info`
- `list`

These rights can apply to any type of resource (in other words, they are not specific to web documents and directories on a web server).

For specific types of resources, such as web documents, you can define specific rights (for example, `http_get` or `http_put`). You need to provide a mapping between the generic rights listed above and the specific rights you define.

The Access Control API includes a mapping between generic and specific rights for defining rights in terms of access over the HTTP protocol. This mapping is `http_generic`.

Chapter

3

Property Lists

The Netscape web servers use property lists to store name-value pairs. The ACL API uses these name-value pairs to determine the access rights of a user to resources protected by access control.

Working with Property Lists

Some of the API functions require arguments that are property lists. For example, the `LASEvalFunc_t` type definition specifies that you can pass property lists for the subject and resource as arguments.

A *property list* is a set of name-value pairs that are accessible by name and by index. The values in property lists can be arbitrary types of values (not just string values).

A value can also have an associated type handle, which can be used to refer to interfaces providing access to the property value data. (A property value can be used to represent an object instance.)

Basically, an entry in a property list can contain the following types of information:

- Name of the property
- Value of the property (not limited to string values)

- Index of the property
- Type of the property

Property Lists Versus NSAPI pblocks

If you have worked with NSAPI and are familiar with the pblock structure, the property list is an extended version of the pblock.

pblocks store name-value pairs, where the values are strings. The property list can store values of arbitrary types, including handles for other property lists. A value in a property list can also represent an object instance. Each property can have an associated type handle that can be used to reference interfaces that provide access to the actual data values.

One other difference between pblocks and property lists is that in a property list, you can identify a value by an integer index or by name. In pblocks, you specify a name to access its value.

The rest of this section explains how to create, manipulate, and destroy property lists.

Property List Types

The Access Control API uses four different property list types to store values on the server.

Subject

The Server creates a subject property list for each session. The subject property list stores information about the subject requesting a resource. This information can include the user's login name, password, dn, and group. The Server stores any information retrieved by getter functions in the subject property list. It also stores a pointer to the Session structure in the subject property list.

Resource

The Server generates a resource property list for each request for a resource from the subject. The Server stores a pointer to the Request structure in the resource property list. There may be many requests during each session.

Auth_info

Information from the authentication block of the ACL is transferred into the auth_info property list. The auth_info property list points to the authentication block for the attribute being evaluated. It stores any name-value pairs which are present in the authentication block, typically method name and database type.

Note You should always use the `ACL_AuthInfoGetDbname`, `ACL_AuthInfoGetDbType`, and `ACL_AuthInfoGetMethod` functions to retrieve database/method information from auth_info Plists rather than using Plist functions. If the auth_info is NULL (there is no authenticate block for the attribute), using the `ACL_AuthInfoGetXYZ` functions will return the proper default values.

Global_auth

The global_auth property list holds authentication information for all attributes. Use it to verify authentication. For example, during an evaluation of the groups attribute, you may want to look up the auth_info for the user attribute; you could use the global_auth property list for this purpose.

Creating a New Property List

To create a new property list, call the `PListNew()` function or the `PListCreate()` function:

- Call `PListCreate()` if you want to reserve a range of indices for properties.
- In both functions, you can specify a memory pool that you want used for allocating values.

When you are done working with the property list, free it by calling the `PListDestroy()` function.

For information on these functions, see Chapter 9, “Functions and Function Typedefs.”

Adding and Removing Properties

You can add properties to the property list in one of two ways:

- Call the `PListInitProp()` function, which adds the property name to the list and assigns a value to that name (and optionally assigns a type to that name).
- Call the following functions individually:
  - Call the `PListDefProp()` function to add a property name to the list.
  - Call the `PListSetValue()` function to assign a value (and, optionally, a type) to that property.

To remove a property from the list, call the `PListDeleteProp()` function.

For information on these functions, see Chapter 9, "Functions and Function Typedefs."

Finding Properties in the Property List

You can find values in the property list by name or by index:

- To get the value of a property by its name, call the `PListFindValue()` function.
- To change the value of a property by its name, call the `PListAssignValue()` function.
- To get the value of a property by its index, call the `PListGetValue()` function.
- To change the value of a property by its index, call the `PListSetValue()` function.
- To change the type of a property by its index, call the `PListSetType()` function.

For information on these functions, see Chapter 9, "Functions and Function Typedefs."

Chapter

4

How the Standard Attributes Work

This chapter explains how the standard attributes are authenticated:

- Defines the standard attributes that can be included in ACLs.
- Explains how ACLs are evaluated.
- Describes the standard evaluation functions which compare the standard attributes against values in a database.
- Describes the standard attribute getter functions that authenticate the standard attributes.

  Attribute getter functions can also check the validity of the attribute. For example, the standard `isvalid-password` attribute checks the password entered by the user against the password stored with the user's entry.
- Explains certificate based authentication.

Standard Attribute Definitions

The following table defines the standard attributes used by the ACL API:

Note You can use the index number of the attribute to reference its value.

Attributes	Index	Description
ACL_ATTR_RAW_USER_LOGIN	2	Not used
ACL_ATTR_USER For more information, see "Getting the user Attribute" and "Evaluating the user Attribute".	7	Represents the authenticated user's login name, stored on the subject property list.
ACL_ATTR_PASSWORD	8	Represents the authenticated user's login password, stored on the subject property list.
ACL_ATTR_USERDN	9	Represents the authenticated user's LDAP DN on the subject property list.
ACL_ATTR_RAW_USER For more information, see "Getting the raw-user Attribute".	10	Represents the raw (unauthenticated) login name extracted from the request header.
ACL_ATTR_RAW_PASSWORD For more information, see "Getting the raw-pw Attribute".	11	Represents the raw (unauthenticated) password extracted from the request header.
ACL_ATTR_USER_ISMEMBER	12	Represents whether a user is a member of given groups.
ACL_ATTR_DATABASE	13	Used to store database name on the auth_info and global_auth property list.
ACL_ATTR_DBTYPE	14	Used to store database type on the auth_info and global_auth property list.
ACL_ATTR_METHOD	17	Used to store internal method representation on the auth_info and global_auth property list.
ACL_ATTR_AUTH_TYPE	18	Not used
ACL_ATTR_AUTHORIZATION For more information, see "Getting the authorization Attribute".	19	Stores raw authorization information from the request header. It contains the method name (for example, "basic"), user's login name, and password concatenated and encoded.
ACL_ATTR_IP For more information, see "Evaluating the ip Attribute".	23	Represents the IP address of the client.

ACL_ATTR_DNS For more information, see "Evaluating the dns Attribute".	24	Represents the dns of the client.
ACL_ATTR_GROUPS For more information, see "Evaluating the group Attribute".	27	The Group LAS passes the list of groups to the "ismember" getter via this property, set on the subject property list.
ACL_ATTR_IS_VALID_PASSWORD For more information, see "Getting the isvalid-password Attribute".	28	Represents whether a user's login/password is valid.
ACL_ATTR_CERT2USER	29	Stores the user name after the certificate is mapped to a user in the database.
ACL_ATTR_USER_CERT For more information, see "Getting the cert Attribute".	30	Stores the client's certificate.
ACL_ATTR_PROMPT	31	Stores the prompt (if any) in the authenticate block in the ACL on the auth_info and global_auth property list.
ACL_ATTR_USERS_GROUP	33	Not used
ACL_ATTR_SESSION	34	Represents the NSAPI Session structure for the Web Server (stored on the subject property list).
ACL_ATTR_REQUEST	35	Represents the NSAPI Request structure for the Web Server (stored on the resource property list).
ACL_ATTR_ERROR	36	
ACL_ATTR_PROGRAMS	37	Used by the Admin Server to implement distributed administration.
ACL_ATTR_WWW_AUTH_PROMPT	39	Prompts to be sent to the client are stored in this property.
ACL_ATTR_IS_OWNER	41	Represents whether the user is owner of the resource being accessed.
ACL_ATTR_CACHED_USER	42	If you are evaluating the ACLs for a particular user (you know the user already and don't want to prompt for login/password), you should store the user on this property on the subject property list. Whenever, the server sees a new authenticate block for a particular attribute (for example, "user"), it clears those properties from the subject property list before proceeding. The built-in getter for ACL_ATTR_AUTH_USER looks for the "cached-user" property first before looking anywhere else.
ACL_ATTR_USER_EXISTS	43	Result of a check for the user in the database.

Note that the following defined names are not attributes, although they appear listed among the attributes in the header file:

- `ACL_ATTR_GROUP`
- `ACL_ATTR_AUTH_USER`
- `ACL_ATTR_AUTH_TYPE`
- `ACL_ATTR_AUTH_DB`
- `ACL_ATTR_AUTH_PASSWORD`
- `ACL_ATTR_DBNAME`
- `ACL_ATTR_DATABASE_URL`
- `ACL_ATTR_PARSEFN`
- `ACL_ATTR_ATTRIBUTE`
- `ACL_ATTR_GETTERFN`
- `ACL_ATTR_MODULE`
- `ACL_ATTR_MODULEFUNC`
- `ACL_ATTR_TIME`
- `ACL_ATTR_ACCEL_AUTH`
- `ACL_ATTR_OWNER`

How ACLs are Evaluated

An evaluation function:

1. Allocates memory for error frames. If any errors occur during this process, the evaluation functions uses the `NSErr_t * errp` parameter to log the errors.
2. Creates a new ACL evaluation context.
3. Sets the ACL to the ACL associated with the cached entry.

4. Creates a new subject property list.
5. Sets the `ip` property (`ACL_ATTR_IP`) in the subject property list to the client's IP address.
6. Gets the `authorization` header.
7. Sets the `accel-authorization` property (`ACL_ATTR_ACCEL_AUTH`) in the subject property list to the `authorization` header.
8. Sets the subject property list in the ACL evaluation context.
9. Creates a new resource property list.
10. Sets the resource property list in the ACL evaluation context.
11. Tests the ACL evaluation context.
12. Frees the resource property list.
13. Frees the subject property list.
14. Frees the ACL evaluation context.
15. Frees the error frame.

The Standard Evaluation Functions

This section describes the evaluation functions for each of the standard attributes. Evaluation functions are responsible for authorization, that is, for comparing attribute values against authorization ACEs once those values are retrieved and validated. Evaluation functions do not depend on method and database, which are specific to authentication and attribute getter functions.

Evaluating the user Attribute

The standard evaluation function for the `user` attribute:

1. Verifies that the attribute in the ACL expression is `user` and that the comparator is either equals or not equals.

If the attribute is not `user` or if the comparator is not `=` or `!=`, the evaluation function returns `LAS_EVAL_INVALID`.

2. Checks to see if the attribute pattern from the ACL expression is `anyone`. (The `"anyone"` means "any unauthenticated user," so this step precedes authentication.)

   - If the ACL expression is `user = "anyone"`, the evaluation function returns `LAS_EVAL_TRUE`.
   - If the ACL expression is `user != "anyone"`, the evaluation function returns `LAS_EVAL_FALSE`.

3. Gets the value of the `user` attribute (`ACL_ATTR_USER`). See "Getting the user Attribute" for details.

   If the attribute getter function for the `user` attribute returns a value other than `LAS_EVAL_TRUE`, the evaluation function returns that value.

4. Checks to see if the attribute pattern from the ACL expression is `all`. (The `"all"` means "all authenticated users," so this step follows authentication.)

   - If the ACL expression is `user = "all"`, the evaluation function returns `LAS_EVAL_TRUE`.
   - If the ACL expression is `user != "all"`, the evaluation function returns `LAS_EVAL_FALSE`.

5. If the attribute pattern matches the value `owner`, gets the value of the `is-owner` attribute (`ACL_ATTR_IS_OWNER`). See "Getting the is-owner Attribute" for details.

6. Compares the value of the `user` attribute against the attribute pattern:

   - If they match and the comparator in the ACL expression is equals (=), returns `LAS_EVAL_TRUE`.
   - If they don't match and the comparator in the ACL expression is equals (=), returns `LAS_EVAL_FALSE`.
   - If they don't match and the comparator in the ACL expression is not equals (!=), returns `LAS_EVAL_TRUE`.

- If they match and the comparator in the ACL expression is not equals (!=), returns LAS_EVAL_FALSE.

Evaluating the group Attribute

The standard evaluation function for the `group` attribute:

1. Verifies that the attribute in the ACL expression is `group` and that the comparator is either equals or not equals.

   If the attribute is not `group` or if the comparator is not `=` or `!=`, the evaluation function returns LAS_EVAL_INVALID.

2. Checks to see if the attribute pattern from the ACL expression is `anyone`.

   - If the ACL expression is `group = "anyone"`, the evaluation function returns LAS_EVAL_TRUE.
   - If the ACL expression is `group != "anyone"`, the evaluation function returns LAS_EVAL_FALSE.

3. Gets the value of the `user` attribute (ACL_ATTR_USER). See "Getting the user Attribute" for details.

   If the attribute getter function for the `user` attribute returns a value other than LAS_EVAL_TRUE, the evaluation function returns that value.

4. Gets the value of the `time` property (by its index, ACL_ATTR_TIME_INDEX) from the resource property list.

   If no value exists for this property, sets the value to the current time.

5. Gets the name of the authentication database (LDAP) from the auth_info property list.

   If the name of the database cannot be retrieved, returns LAS_EVAL_FAIL.

6. Uses the database name to get the connection to the LDAP directory.

7. For each group in the ACL expression, checks the cache to see if the group is in the cache.

User information (such as the groups that a user belongs to) is cached for 2 minutes. If the user information is not cached, this function does the following to get that information:

- Removes the `groups` property (ACL_ATTR_GROUPS) from the subject property list.
- Sets the `groups` property to the groups listed in the ACL expression.
- Removes the `user-ismember` property (ACL_ATTR_USER_ISMEMBER) from the subject property list.
- Gets the value of the `user-ismember` attribute. See "Getting the user-ismember Attribute" for details.
- Removes the `groups` property (ACL_ATTR_GROUPS) from the subject property list.

8. Depending on the value returned by the `user-ismember` attribute getter function, the evaluation function does one of the following:

   - If LAS_EVAL_TRUE, checks to see if the `users-group` property (ACL_ATTR_USERS_GROUP) is set in the subject property list. If so, the evaluation function updates the cache with this value and removes this property from the subject property list. Then, the evaluation function returns LAS_EVAL_TRUE if the comparator is equals (=) or LAS_EVAL_FALSE if the comparator is not equals (!=).
   - If LAS_EVAL_FALSE, returns LAS_EVAL_FALSE if the comparator is equals (=) or LAS_EVAL_TRUE if the comparator is not equals (!=).
   - If the attribute getter function returns any other value, this evaluation function returns with that value.

Evaluating the dns Attribute

The standard evaluation function for the `dns` attribute:

1. Verifies that the attribute in the ACL expression is `dns` or `dnsalias` and that the comparator is either equals or not equals.

   If the attribute is not `dns` or `dnsalias` and if the comparator is not `=` or `!=`, the evaluation function returns `LAS_EVAL_INVALID`.

2. Gets the value of the `dns` attribute (`ACL_ATTR_DNS`). See "Getting the dns Attribute" for details.

   If the attribute getter function for the `dns` attribute does not return `LAS_EVAL_TRUE`, the evaluation function returns `LAS_EVAL_FAIL`.

3. Compares the host name of the client against the attribute pattern in the ACL expression.

   - If they match and the comparator in the ACL expression is equals (=), returns `LAS_EVAL_TRUE`.
   - If they don't match and the comparator in the ACL expression is equals (=), returns `LAS_EVAL_FALSE`.
   - If they don't match and the comparator in the ACL expression is not equals (!=), returns `LAS_EVAL_TRUE`.
   - If they match and the comparator in the ACL expression is not equals (!=), returns `LAS_EVAL_FALSE`.

Evaluating the ip Attribute

The standard evaluation function for the `ip` attribute:

1. Verifies that the attribute in the ACL expression is `ip` and that the comparator is either equals or not equals.

   If the attribute is not `ip` and if the comparator is not `=` or `!=`, the evaluation function returns `LAS_EVAL_INVALID`.

2. Gets the value of the `ip` attribute (`ACL_ATTR_IP`). See "Getting the ip Attribute" for details.

   If the attribute getter function for the `ip` attribute does not return `LAS_EVAL_TRUE`, the evaluation function returns `LAS_EVAL_FAIL`.

3. Compares the IP address of the client against the attribute pattern in the ACL expression.

   - If they match and the comparator in the ACL expression is equals (=), returns `LAS_EVAL_TRUE`.
   - If they don't match and the comparator in the ACL expression is equals (=), returns `LAS_EVAL_FALSE`.
   - If they don't match and the comparator in the ACL expression is not equals (!=), returns `LAS_EVAL_TRUE`.
   - If they match and the comparator in the ACL expression is not equals (!=), returns `LAS_EVAL_FALSE`.

The Standard Attribute Getter Functions

This section describes the attribute getter functions for each of the standard attributes. Attribute getter functions are responsible for authentication. These standard attribute getter functions may be specific to standard methods (`basic` or `ssl`) or the standard database type (`LDAP`). For an example of how to write your own attribute getter functions see "Getting the Value of the Referer Attribute".

How ACL_GetAttribute() Works

The `ACL_GetAttribute()` function gets the value of a specified attribute as follows:

1. If the subject property list is `NULL`, returns `LAS_EVAL_FAIL`.
2. Checks the subject property list for the value of the attribute.

3. If the subject property list contains the attribute and its value, passes the value of the attribute back through the `val` argument and returns `LAS_EVAL_TRUE`.

4. If the subject property list does not contain the attribute and its value, gets the authentication method and database type information from the auth_info property list.

5. Retrieves a list of the attribute getter functions for this attribute.

6. Iterates through this list, finding and invoking the attribute getter functions registered under the same method and database type that are specified in the auth_info property list.

7. If an attribute getter function returns `LAS_EVAL_TRUE`, `ACL_GetAttribute()` verifies that the attribute has been added to the subject property list as a property:

   - If so, `ACL_GetAttribute()` returns `LAS_EVAL_TRUE` and passes the value of the attribute back as the `val` argument.

   - If, however, the attribute is not part of the subject property list, `ACL_GetAttribute()` returns `LAS_EVAL_FAIL`.

8. If an attribute getter function returns `LAS_EVAL_FAIL` or `LAS_EVAL_INVALID`, `ACL_GetAttribute()` returns the same value.

9. If an attribute getter function returns `LAS_EVAL_DECLINE`, `ACL_GetAttribute()` repeats this process for the next matching attribute getter function in the list.

10. If any errors occur during this process, the attribute getter functions uses the `NSErr_t * errp` parameter to log the errors.

Getting the user Attribute

Basic Method, Any Database

The standard attribute getter function that is registered for getting the `user` attribute, using the `basic` method and any database:

1. Gets the value of the `raw-user` attribute (`ACL_ATTR_RAW_USER`). See "Getting the raw-user Attribute" for details.

   If the `raw-user` attribute has no value assigned to it, returns `LAS_EVAL_FAIL`.

2. Gets the value of the `raw-pw` attribute (`ACL_ATTR_RAW_PASSWORD`). See "Getting the raw-pw Attribute" for details.

   Note that the `raw-pw` attribute getter function is the same function as the `raw-user` attribute getter function.

3. Gets the value of the `request` property (by its index, `ACL_ATTR_REQUEST_INDEX`) in the resource property list.

   If unable to get the value of the `request` property, returns `LAS_EVAL_FAIL`.

4. Deletes any existing `isvalid-password` property (`ACL_ATTR_IS_VALID_PASSWORD`) from the subject property list.

5. Gets the value of the `isvalid-password` attribute. See "Getting the isvalid-password Attribute" for details.

6. If the `isvalid-password` attribute getter function returns `LAS_EVAL_FALSE`, the `user` attribute getter function does the following:

   - Gets the value of the `session` property (by its index, `ACL_ATTR_SESSION_INDEX`) from the subject property list.
   - Gets the value of the `prompt` property (by its index, `ACL_ATTR_PROMPT_INDEX`) from the auth_info property list.
   - Adds the `www-auth-prompt` property to the resource property list, assigning it the value of the `prompt` property.
   - Returns the value `LAS_EVAL_NEED_MORE_INFO`.

7. In the `rq->vars` pblock, inserts the following name/value pairs:
   - Sets `auth-type` (`ACL_ATTR_AUTH_TYPE`) to basic.
   - Sets `auth-user` (`ACL_ATTR_AUTH_USER`) to the username.
   - Sets `auth-password` (`ACL_ATTR_AUTH_PASSWORD`) to the password.
   - Sets `userdn` (`ACL_ATTR_USERDN`) to the user's DN.
8. Sets the value of the `user` property of the subject property list to the username.
9. Returns `LAS_EVAL_TRUE`.

SSL Method, Any Database

The standard attribute getter function that is registered for getting the `user` attribute, using the `ssl` method and any database:

1. Gets the value of the `cert` attribute (`ACL_ATTR_USER_CERT`). See "Getting the cert Attribute" for details.
2. Gets the value of the `cert2user` attribute (`ACL_ATTR_CERT2USER`). See "Getting the cert2user Attribute" for details.

   If the `cert2user` getter function fails (if it does not return `LAS_EVAL_TRUE`), the `user` attribute getter function returns with the error code returned by the `cert2user` getter function.

   If the `cert2user` attribute has no value assigned to it, returns `LAS_EVAL_FAIL`.
3. Sets the value of the `user` property of the subject property list to the username.
4. Returns `LAS_EVAL_TRUE`.

Getting the isvalid-password Attribute

Basic Method, LDAP Database

The standard attribute getter function that is registered for getting the `isvalid-password` attribute, using the `basic` method and an `LDAP` database:

1. Gets the value of the `raw-user` attribute (`ACL_ATTR_RAW_USER`). See "Getting the raw-user Attribute" for details.

   If the `raw-user` getter function fails (if it does not return `LAS_EVAL_TRUE`), the `isvalid-password` attribute getter function returns with the error code returned by the `raw-user` getter function.

2. Gets the value of the `raw-pw` attribute (`ACL_ATTR_RAW_PASSWORD`). See "Getting the raw-pw Attribute" for details.

   - If the `raw-pw` getter function fails (if it does not return `LAS_EVAL_TRUE`), the `isvalid-password` attribute getter function returns with the error code returned by the `raw-pw` getter function.
   - If the password is `NULL`, authentication fails and the value `LAS_EVAL_FALSE` is returned.

3. Gets the name of the authentication database (`LDAP`) from the auth_info property list.

   If the name of the database cannot be retrieved, returns `LAS_EVAL_FAIL`.

4. Uses the database name to get the connection to the LDAP directory.

5. Gets the value of the `time` property (by its index, `ACL_ATTR_TIME_INDEX`) from the resource property list.

   If no value exists for this property, sets the value to the current time.

6. Checks the cache to see if the user's information is cached.

   User information (such as the username, password, DN, base-64 DER-encoded client certificate, and group of a user) is cached for 2 minutes. If the user information is not cached, this function does the following in order to get that information:

- Initializes a session with the LDAP directory and attempts to bind to the directory. If the bind operation fails, returns LAS_EVAL_FAIL.
- Finds the distinguished name for the user. To find the user's entry in the directory, the directory is searched for an entry where the value of the uid attribute matches the user's name.
- Uses the distinguished name to bind to the database. If the user's password is not valid, LAS_EVAL_FALSE is returned. If an LDAP error occurs, LAS_EVAL_FAIL is returned.
- Caches the username and password information.

7. In the subject property list, sets the isvalid-password property (ACL_ATTR_IS_VALID_PASSWORD) to true and the userdn property (ACL_ATTR_USERDN) to the user's distinguished name.
8. Returns LAS_EVAL_TRUE.

Getting the raw-user Attribute

Basic Method, Any Database

The standard attribute getter function that is registered for getting the raw-user attribute, using the basic method and any database:

1. Gets the value of the authorization attribute (ACL_ATTR_AUTHORIZATION). See "Getting the authorization Attribute" for details.
   - If the authorization getter function fails (if it does not return LAS_EVAL_TRUE), the raw-user attribute getter function returns with the error code returned by the authorization getter function.
   - If the authorization attribute has no value assigned to it, the raw-user attribute getter function returns LAS_EVAL_FAIL.
2. Parses the value of the authorization attribute to get the username and password.

3. Sets the value of the `raw-user` property of the subject property list to the username.

4. Sets the value of the `raw-pw` property of the subject property list to the password.

5. Returns `LAS_EVAL_TRUE`.

Getting the authorization Attribute

Basic Method, Any Database

The standard attribute getter function that is registered for getting the `authorization` attribute, using the `basic` method and any database:

1. Gets the value of the `accel-authorization` property (by its index, `ACL_ATTR_ACL_AUTH_INDEX`) from the subject property list.

   This is the value of the `authorization` header in the HTTP request.

   If this property is not in the subject property list, the attribute getter function gets the value of the `authorization` header in the HTTP request through the headers in the request object.

2. If the authentication header has no value, the attribute getter function does the following:

   - Gets the value of the `session` property (by its index, `ACL_ATTR_SESSION_INDEX`) from the subject property list.
   - Gets the value of the `prompt` property (by its index, `ACL_ATTR_PROMPT_INDEX`) from the auth_info property list.
   - Adds the `www-auth-prompt` property to the resource property list, assigning it the value of the `prompt` property.
   - Returns the value `LAS_EVAL_NEED_MORE_INFO`.

3. If the authentication header has a value, the attribute getter function does the following:

- Sets the value of the `authorization` property in the subject property list to the value of the `authorization` header.
- Returns the value `LAS_EVAL_TRUE`.

Getting the raw-pw Attribute

Basic Method, Any Database

The attribute getter function for the `raw-pw` attribute (`ACL_ATTR_RAW_PASSWORD`) is the same function as the attribute getter function for the `raw-user` attribute (`ACL_ATTR_RAW_USER`).

For details, see "Getting the raw-user Attribute".

Getting the is-owner Attribute

Any Method, Any Database

The standard attribute getter function that is registered for getting the `is-owner` attribute, using any method and any database:

1. In the subject property list, sets the `is-owner` property (`ACL_ATTR_IS_OWNER`) to `true`.
2. Returns `LAS_EVAL_TRUE`.

Getting the cert Attribute

SSL Method, Any Database

The standard attribute getter function that is registered for getting the `cert` attribute, using the `ssl` method and any database:

1. Gets the value of the `session` property (by its index, `ACL_ATTR_SESSION_INDEX`) from the subject property list.

   If unable to get the value of the `session` property, returns `LAS_EVAL_FAIL`.

2. Gets the value of the `request` property (by its index, `ACL_ATTR_REQUEST_INDEX`) in the resource property list.

   If unable to get the value of the `request` property, returns `LAS_EVAL_FAIL`.

3. If the server does not have SSL enabled, returns `LAS_EVAL_FAIL`.

4. Gets the certificate from the client.

   If the client certificate is not already present in the session, repeats the SSL handshake and gets the client certificate.

   If the client certificate cannot be retrieved, returns `LAS_EVAL_FAIL`.

5. Inserts a base-64 DER-encoded version of the certificate under the name/value pair `auth-cert` in `rq->vars`.

6. Sets the value of the `cert` property (`ACL_ATTR_USER_CERT`) in the subject property list to the client's certificate.

7. Returns `LAS_EVAL_TRUE`.

Getting the cert2user Attribute

SSL Method, LDAP Database

The standard attribute getter function that is registered for getting the `cert2user` attribute, using the `ssl` method and an `LDAP` database:

1. Gets the value of the `session` property (by its index, `ACL_ATTR_SESSION_INDEX`) from the subject property list.

   If unable to get the value of the `session` property, returns `LAS_EVAL_FAIL`.

2. Gets the value of the `request` property (by its index, `ACL_ATTR_REQUEST_INDEX`) in the resource property list.

   If unable to get the value of the `request` property, returns `LAS_EVAL_FAIL`.

3. Gets the name of the authentication database (`LDAP`) from the auth_info property list.

   If the name of the database cannot be retrieved, returns `LAS_EVAL_FAIL`.

4. Uses the database name to get the connection to the LDAP directory.

5. Gets the value of the `time` property (by its index, `ACL_ATTR_TIME_INDEX`) from the resource property list.

   If no value exists for this property, sets the value to the current time.

6. Finds the user entry in the LDAP directory that corresponds to the client certificate.

   This process is described in the *Certificate Mapping Programmer's Guide*.

   If a corresponding user entry cannot be found, returns `LAS_EVAL_FALSE`. If an error occurs during the mapping process, returns `LAS_EVAL_FAIL`.

7. In the `rq->vars` pblock, inserts the following name/value pairs:

   - Sets `auth-type` (`ACL_ATTR_AUTH_TYPE`) to SSL.
   - Sets `auth-user` (`ACL_ATTR_AUTH_USER`) to the username.
   - Sets `auth-db` (`ACL_ATTR_AUTH_DB`) to the name of the database.

- Sets `userdn` (`ACL_ATTR_USERDN`) to the user's DN.

8. Sets the value of the `cert2user` property in the subject property list to the username.

9. Returns `LAS_EVAL_TRUE`.

Getting the user-ismember Attribute

Any Method, LDAP Database

The standard attribute getter function that is registered for getting the `user-ismember` attribute, using any method and an `LDAP` database:

1. Gets the value of the `user` attribute (`ACL_ATTR_USER`). See "Getting the user Attribute" for details.

   If the `user` attribute getter function does not return `LAS_EVAL_TRUE`, the `user-ismember` attribute getter function returns `LAS_EVAL_FAIL`.

2. Gets the value of the `groups` attribute (`ACL_ATTR_GROUPS`).

   If the `groups` attribute getter function does not return `LAS_EVAL_TRUE`, the `user-ismember` attribute getter function returns `LAS_EVAL_FAIL`.

3. Gets the name of the authentication database (`LDAP`) from the auth_info property list.

4. Uses the database name to get the connection to the LDAP directory.

5. Given the group name, searches for the group entry in the LDAP directory.

   If an error occurs, returns `LAS_EVAL_FAIL`.

6. Finds the user entry in the LDAP directory.

7. Performs a base search to determine if the group entry has the attribute `uniquemember = <user_dn>` or `member = <user_dn>`.

8. If the user is a member of the specified group, this function does the following:

- Sets the value of the `users-group` property in the subject property list to the group name.
- Sets the value of the `user-ismember` property in the subject property list to `true`.
- Returns LAS_EVAL_TRUE.

9. If the user is not a member of the specified group, this function does the following:
   - Sets the value of the `user-ismember` property in the subject property list to `false`.
   - Returns LAS_EVAL_FALSE.

Getting the dns Attribute

Any Method, Any Database

The standard attribute getter function that is registered for getting the `dns` attribute, using any method and any database:

1. Gets the value of the `session` property (by its index, ACL_ATTR_SESSION_INDEX) from the subject property list, and attempts to get the host name of the client from the session.
2. If it can't get the value of the `session` property, gets the value of the `ip` property (by its index, ACL_ATTR_IP_INDEX) from the subject property list, and attempts to get the host name of the client from its IP address.
3. If the host name can't be retrieved, uses `unknown` for the host name.
4. Sets the value of the `dns` property in the subject property list to the host name.

   If the value can't be set, returns LAS_EVAL_FAIL.
5. Returns LAS_EVAL_TRUE.

Getting the ip Attribute

Any Method, Any Database

The standard attribute getter function that is registered for getting the `ip` attribute, using any method and any database:

1. Gets the value of the `session` property (by its index, `ACL_ATTR_SESSION_INDEX`) from the subject property list.

   If unable to get the value of the `session` property, returns `LAS_EVAL_FAIL`.

2. Gets the IP address of the client from the session.

3. Sets the value of the `ip` property in the subject property list to the IP address of the client.

   If the value can't be set, returns `LAS_EVAL_FAIL`.

4. Returns `LAS_EVAL_TRUE`.

Certificate-Based Authentication

When a user authenticates to a Netscape server by sending a client certificate to the server, the server uses information in the certificate to search the user directory for the user's entry.

You can configure some parts of this process by editing the file *`server root`*`/userdb/certmap.conf`. This file specifies:

- How the server searches the directory for the user's entry
- Whether or not the server goes through an additional step of verifying that the user's certificate matches the certificate presented to the server

You can modify this "certificate to directory entry" process programmatically. Netscape servers include a set of API functions that allow you to control this process. You can write your own functions to customize how certificate subject

entries are found in the directory. However, the Certificate Mapping API can only be used for certificates and/or LDAP. If you are not using LDAP, you can still use some functions from the API to extract info from a certificate.

Note To use the Certificate Mapping API, you msut have the Directory SDK. You can download this SDK from the Netscape DevEdge site at `http://developer.netscape.com/`.

For more information on the Certificate Mapping API, see the *Certificate-Mapping Programmer's Guide* at `http://developer.netscape.com/library/documentation/certificate/index.htm`.

LDAP Databases

LDAP (Lightweight Directory Access Protocol) is a protocol for accessing and managing directory services. The Enterprise Server uses LDAP directories as its standard database. A directory consists of entries containing descriptive information (for example, entries describing people or network resources). For more information on LDAP directories and the LDAP API see the *Netscape Directory SDK Programmer's Guide* at `http://developer.netscape.com/library/documentation/dirsdk/capi/contents.htm`.

Mapping Client Certificates to LDAP

When the server gets a request from a client, it asks for the client's certificate before proceeding. Netscape clients, such as Netscape Navigator and Netscape Communicator, send the client certificate to the server (with or without prompting the user, depending on the browser's security configuration). The server tries to match the certificate authority listed in the certificate to a trusted certificate authority listed in the Administration Server. If there isn't a match, some servers end the connection and some perform a different operation based on the failed match. If there is a match, the server continues processing the request.

After the server checks that the certificate's issuer is trusted, the server goes through three steps to map the certificate to an LDAP entry:

1. It maps the certificate subject to a branch point in the LDAP directory.

2. It searches the LDAP directory for entries that match the information about the certificate subject.

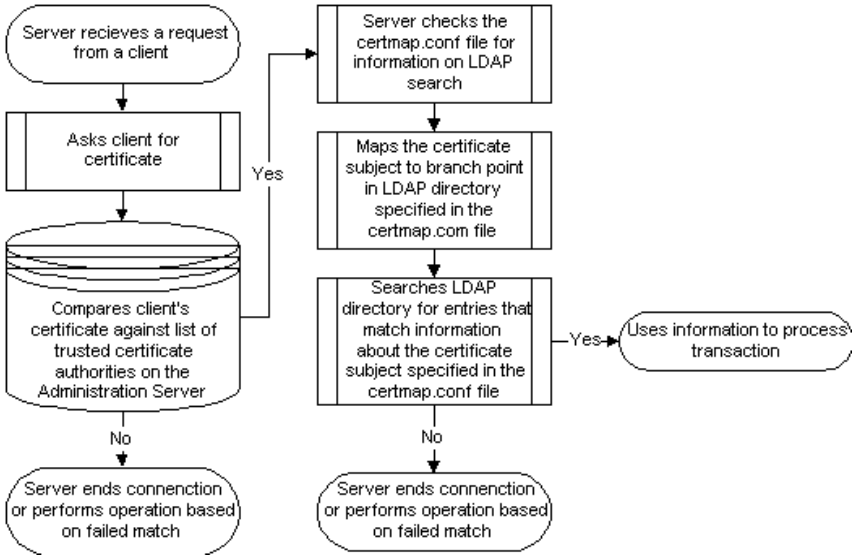
3. It optionally verifies the client certificate with one in the LDAP directory.

The server uses the `certmap.conf` file to determine where to start, and how to do, the LDAP search. The mapping file tells the server what values to take from the client certificate (such as the owner's name, email address, and so on). The server uses these values to search for a user entry in the LDAP directory. The Certificate Mapping API works independently from the ACLs and method.

Once the server knows where to start its search and what it needs to search for (step 1), it performs the search in the LDAP directory (step 2). If it finds more than one matching entry and the mapping is not set to verify the certificate, the search fails. Some servers end the transaction at this point.

When the server finds multiple matching entries in the directory, the server can optionally verify the client's certificate by comparing it with certificates for the matching entries in the LDAP directory (step 3). If the client certificate doesn't match any certificates in the matching entries or if the matching entries don't contain certificates, the certificate mapping fails. At this point, some servers end the transaction with the client; others perform an action based on the failed match.

After the server finds a matching entry and certificate in the LDAP directory, it can use that information to process the transaction.



Using the certmap.conf File

The certificate mapping file determines how a server should look up a user entry in the LDAP directory. You edit this file and add entries to match the organization of your LDAP directory and to list the certificates you want your users to have. specifically, the mapping file defines:

- Where in the LDAP tree the server should begin its search
- What certificate attributes the server should use as search criteria when searching for the entry in the LDAP directory
- Whether the server goes through an additional verification process

The certificate mapping file is located at *server-root*`/userdb/certmap.conf`. The file contains one or more named mappings, each applying to a different certificate authority. A mapping has the following syntax:

```
certmap name issuerDN
name:property [value]
```

The first line specifies a name for the mapping. The *name* is arbitrary; you can define it to be whatever you want. However, *issuerDN* must exaclty match the distinguished of the certificate authority who issued the client certificate. For more information on the Certificate-Mapping API see the *Certificate Mapping Programmer's Guide*.

Chapter

5

Defining Attributes for Authentication

The Access Control API allows you to define your own attributes to use when authenticating users. (For example, you might want to authenticate users based on email address or on the URL that referred users to the resource.) To configure the server to use your attributes for authentication, you need to define your own Loadable Authentication Service (LAS).

To define an LAS, you:

1. Define a getter function to get the value of the attribute from the subject requesting access. (See "Defining a Function to Get the Attribute Value".)
2. Define an evaluation function that evaluates that value against the authorization statements in an ACL. (See "Defining a Function to Evaluate the Attribute Value".)
3. Define a function for flushing the cache, if you plan to cache the attribute values that you retrieve. (See "Defining a Function to Flush the Cache".)
4. Define an initialization function that registers the LAS and the functions you've defined. (See "Writing the Initialization Function".)
5. Configure the server to run the initialization function on startup. (See "Configuring the Server".)

For example, suppose you want to authenticate users based on the URL that referred the users to the resource. You need to:

1. Define a function that gets the HTTP referer header from the client request.
2. Define a function that the server will use to determine if the value of the referer attribute for the user matches the authorization statements in an ACL (for example, “referer = *.netscape.com”).
3. Define a function for flushing these referer attributes from the cache, if you plan to cache these values.
4. Define an initialization function that registers the LAS for the referer attribute and that registers your functions.
5. Configure the server to run the initialization function on startup.

To see the complete example of defining an LAS for the referer attribute, see “LAS Code for the HTTP Referer Attribute”.

This chapter also describes how you can call LDAP API functions to get or evaluate attribute values (for details, see “Using an LDAP Directory for Authentication”).

Defining a Function to Evaluate the Attribute Value

Once the server has the attribute value for the user, the server needs to compare this value against the ACL statements. You write an evaluation function that performs this comparison.

The evaluation function must be defined consistently with the type definition `LASEvalFunc_t` in `acldef.h`. This function should:

- Call the `ACL_GetAttribute()` function to get the value of the attribute. The `ACL_GetAttribute()` function calls the attribute getter function that you registered for this attribute (see “Defining a Function to Get the Attribute Value”).
- Compare this value against the value supplied by the `attr_pattern` argument. This argument represents the value in the ACL statement used for comparison (for example, in the statement referer = “http://home.netscape.com”, attr_pattern is “http://home.netscape.com”).

- Return one of the following values:
  - If the value of the attribute for the user matches the value in the ACL file, return `LAS_EVAL_TRUE`.
  - If the values do not match, return `LAS_EVAL_FALSE`.
  - If an invalid parameter was passed into your function (causing the evaluation to fail), return `LAS_EVAL_INVALID`.
  - If an error occurs (for example, if you cannot allocate memory to perform the comparison), return `LAS_EVAL_FAIL`.
- If you want to cache the retrieved values, use the `cookie` argument to pass the cached data to subsequent function calls.

  If your function caches data, define a function that flushes the cache. See "Defining a Function to Flush the Cache" for more information.

For example, the following evaluation function checks the value of the referer attribute for the user against the ACL for a resource.

```
int las_ref_eval(NSErr_t *errp, char *attr_name, CmpOp_t comparator,
   char *attr_pattern, ACLCachable_t *cachable, void **cookie,
   PList_t subject, PList_t resource, PList_t auth_info,
   PList_t global_auth)
{
   char *referer;
   int rv;
   int patlen, reflen;

   /* Get the value of the "referer" attribute. (This calls the
   attribute getter function you registered for this attribute.) */

   rv = ACL_GetAttribute(errp, LASREF_ATTRNAME, (void **)&referer,
      subject, resource, auth_info, global_auth);

   if (rv == LAS_EVAL_TRUE) {

      /* In the ACL file, if the attribute is compared against
      a pattern, check the operator used for comparison. For
      example, if the ACL file contains the statement:
         "referer = http://home.netscape.com",
      the comparator is CMP_OP_EQ and the attr_pattern is
      "http://home.netscape.com" */

      if (attr_pattern) {

         /* Based on the operator used for comparison, use a
         different mechanism for comparison. */
```

```
    switch (comparator) {

       /* If the ACL file contains “referer = attr_pattern”,
       use a case-insensitive comparison. */

       case CMP_OP_EQ:

          rv = strcasecmp(referer, attr_pattern) ? LAS_EVAL_FALSE
             : LAS_EVAL_TRUE;
          break;

       /* If the ACL file contains “referer != attr_pattern”,
       use a case-insensitive comparison. */

       case CMP_OP_NE:

          rv = strcasecmp(referer, attr_pattern) ? LAS_EVAL_TRUE
             : LAS_EVAL_FALSE;
          break;

       /* If the ACL file contains “referer < attr_pattern”
       or “referer <= attr_pattern, use a case-insensitive
       comparison to compare the referer against the prefix
       of the attr_pattern. */

       case CMP_OP_LT:
       case CMP_OP_LE:

          reflen = strlen(referer);
          patlen = strlen(attr_pattern);

          rv = ((reflen > patlen) ||
             strncasecmp(referer, attr_pattern, reflen))
                ? LAS_EVAL_FALSE : LAS_EVAL_TRUE;

          break;

       /* If the ACL file contains “referer > attr_pattern”
       or “referer >= attr_pattern, use a case-insensitive
       comparison to compare the attr_pattern against the
       prefix of the referer. */

       case CMP_OP_GT:
       case CMP_OP_GE:

          reflen = strlen(referer);
          patlen = strlen(attr_pattern);

          rv = ((patlen > reflen) ||
             strncasecmp(referer, attr_pattern, patlen))
                ? LAS_EVAL_FALSE : LAS_EVAL_TRUE;

          break;
    }
}
```

```
        return rv;
    }
    return LAS_EVAL_FAIL;
}
```

Defining a Function to Get the Attribute Value

To define how the server gets the value of the attribute for a particular user, you write one or more *attribute getter* functions. Attribute getter functions must be defined consistently with the type definition `ACLAttrGetterFn_t` in `acldef.h`.

The function must return one of the following values:

- If your function successfully gets the attribute value for the user, return `LAS_EVAL_TRUE`.
- If your function cannot get the attribute value for the user, return `LAS_EVAL_FALSE`.
- If an invalid parameter was passed into your function (causing your function to fail), return `LAS_EVAL_INVALID`.
- If an error occurs (for example, if you cannot allocate memory to get the attribute value), return `LAS_EVAL_FAIL`.
- The function can also return `LAS_EVAL_DECLINE` to indicate that it "declines" to get the attribute value. You can use this in situations where you've defined several attribute getter functions for the same attribute. Upon receiving an `LAS_EVAL_DECLINE`, the server attempts to use a different function to get the attribute value.

For example, suppose you wanted to set up authentication based on the URL that referred the subject to a resource (a "referer" attribute). You need to define an attribute getter function that gets this HTTP referer header from the incoming client request.

The following attribute getter function gets the value of the HTTP referer header from the subject accessing the resource. The function puts this value in the subject property list.

```
/* Attribute getter function for the referer attribute */

int las_ref_get(NSErr_t *errp, PList_t subject, PList_t resource,
      PList_t auth_info, PList_t global_auth, void *arg)
{
   Request *rq;
   char *referer;
   int rv;

   /* Get pointer to NSAPI Request structure from resource
   property list. ACL_ATTR_REQUEST_INDEX is the index of the
   property containing the pointer to the NSAPI Request structure. */

   rv = PListGetValue
         (resource, ACL_ATTR_REQUEST_INDEX, (void **)&rq, NULL);

   if (rv < 0) {
      ereport(LOG_FAILURE, "las-ref-get missing request");
      return LAS_EVAL_FAIL;
   }

   /* Try to get the HTTP referer header */
   referer = pblock_findval("referer", rq->headers);

   if (referer) {

      /* Add the referer to the subject property list */
      PListInitProp(subject, 0, "referer", STRDUP(referer), NULL);

      /* Log referer string for debug purposes */
      ereport(LOG_INFORM, "las-ref-get: referer=%s", referer);

      /* Successfully retrieved referer */
      return LAS_EVAL_TRUE;
   }

   /* Referer not found */
   return LAS_EVAL_FAIL;
}
```

Defining a Function to Flush the Cache

If your attribute getter function caches attribute values, you need to write a function that flushes this cache. The function to flush the cache must be defined consistently with the type definition `LASFlushFunc_t` in `acldef.h`.

If you do not cache attribute values, you can define a function that does nothing. For example:

```
/* LASMyFlushFunction
 * Deallocates any memory previously allocated by the LAS
 */

void LASMyFlushFunction(void **las_cookie)
{
   return; /* do nothing */
}
```

Writing the Initialization Function

To initialize your LAS, you write a function that is defined consistently with the type definition `AclModuleInitFunc` in `acldef.h`. This function should:

- Call the `ACL_LasRegister()` function to register the LAS, its evaluation function, and the associated function for flushing the cache (if any).
- Call the `ACL_AttrGetterRegister()` function to register your attribute getter function(s).
- Return zero for success and non-zero for failure.

When you register your attribute getter function, you specify an authentication method and database type associated with the function. This instructs the web server to use this authentication method and database type when getting the value of this attribute. If your function is not associated with a specific method or database type, you can use `ACL_METHOD_ANY` and `ACL_DBTYPE_ANY` (respectively) as the method and database type.

You can register multiple attribute getter functions for a single attribute by registering each attribute getter function under a different method and database type. When the attribute value needs to be retrieved, the server attempts to find the attribute getter function that uses the same authentication method and database type as required by the ACL.

For more information on methods and database types, see Chapter 6, “Defining Methods and Database Types”.

For example, the following section of code registers a new LAS for the attribute `MyAttributeName`. As part of the process of registering the LAS, the example also registers `MyEvalFunction()` as the evaluation function and `MyFlushFunction()` as the cache flushing function.

The example also registers an attribute getter function to get the value of the `MyAttributeName` attribute. The attribute getter function is named `MyAttrGetter()` and is independent of method and database type.

```
NSAPI_PUBLIC int MyLASModuleInit(NSErr_t *errp)
{
   int rv;

   /* ACL_AttrGetterRegister and ACL_LasRegister both return 0
    * if successful. This function does the same.
    */

   rv = ACL_AttrGetterRegister(errp, “MyAttributeName”, MyAttrGetter,
      ACL_METHOD_ANY, ACL_DBTYPE_ANY, ACL_AT_FRONT, NULL);

   if (rv != 0) return rv;

   rv = ACL_LasRegister(errp, “MyAttributeName”, MyEvalFunction,
      MyFlushFunction);

   return rv;
}
```

Configuring the Server

After you write your initialization function, you need to configure your server to call this function on startup. To do so, add the following lines to the initialization section in your `obj.conf` file:

```
Init fn="load-modules" funcs="nameOfYourInitFunction"
shlib="pathToYourSharedObjectOrDLL"

Init fn=acl-register-module module="nameOfYourModule"
func="nameOfYourInitFunction"
```

Do *not* insert any line breaks or carriage returns within a statement. For the module name, provide a name such as the simple name of the DLL or `.so` file. This name is used in error messages.

For example, add the following lines to register your LAS if your initialization function is named `MyLASModuleInit()` and is defined in `/usr/netscape/suitespot/lib/lasmyattr.so`.

```
Init fn="load-modules" funcs="MyLASModuleInit" shlib="/usr/netscape/
suitespot/lib/lasmyattr.so"

Init fn=acl-register-module module="lasmyattr" func="MyLASModuleInit"
```

Passing Additional Arguments During Registration

To pass additional arguments to your module init function (for example, you might need to get information from a configuration file), you must initialize your module using an NSAPI Init directive. This directive can pass a pblock constructed from the Init directive arguments to the function specified in the "fn" argument. This allows you to call the `ACL_ModuleRegister()` function from your new init function and pass your additional arguments. The new code in the `obj.conf` file should read as follows:

```
Init fn="load-modules" funcs="MyLASModuleInit" shlib="/usr/netscape/
suitespot/lib/lasmyattr.so"

Init fn="MyLASModuleInit" prop1=val1 prop2=val2
```

Note that you should not insert any line breaks or carriage returns within each statement. Your new function should conform to NSAPI init function standards instead of ACL module init function standards. For information on NSAPI see the *NSAPI Programmer's Guide* at `http://developer.netscape.com/library/documentation/enterprise/nsapi/index.htm`.

Using an LDAP Directory for Authentication

If you are using an LDAP directory for your authentication database, you can define LAS functions based on the attributes of entries in the directory. For example, if you have defined an LAS that authenticates based on email address (the user's email is an attribute), your evaluation function can search the directory for a matching entry with the same email address.

You can call the `ACL_LDAPDatabaseHandle()` function to get the LDAP database handle. You can then use this handle in subsequent LDAP API calls to search the directory.

For example, the following evaluation function checks the value of the email attribute.

```
#include <ldap.h>
#include <nsacl/aclapi.h>

#define ACL_ATTR_EMAIL "email"

...
```

```
   /* Get the database handle of the LDAP directory. */
   rv = ACL_LDAPDatabaseHandle(errp, NULL, &ld, &basedn);

   if (rv != LAS_EVAL_TRUE) {
      fprintf(stderr, “unable to get LDAP handle\n”);
      return rv;
   }

   ...

   /* Now call LDAP API functions, passing ld as the argument
   for the handle to the database. */

   ...
}
```

Defining LDAP Directories

The `dbswitch.conf` file is used to define logical database names with database type and URL. ACLs always refer to databases by their logical names. This allows you to change the location and/or type of the database just by modifying the `dbswitch.conf` file. The `dbswitch.conf` file format is as follows:

```
directory <database logical name> <dbtype:url>
```

For example:

```
directory Users ldap://F:\Netscape\Suitespot/userdb/ldap/users
```

Note that you should not insert any line breaks or carriage returns within the statement.

For more information on the `dbswitch.conf` file, see Chapter 6.

Chapter

6

Defining Methods and Database Types

When you specify an authentication statement in an ACL file, you can include the authentication method and authentication database used.

For example:

```
acl "path=/usr/netscape/suitespot/docs/index.html";

authenticate (user,group) {
   method = "SSL";
   database = "default";
};

deny (all)
   user = "mozilla";
```

This set of ACL statements specifies that the user named mozilla is denied access to a page on the server. The user authenticates to the server using SSL. The default authentication database is used.

When the server starts up, it automatically registers the `basic` and `ssl` methods (which represent basic HTTP authentication and authentication through SSL) and the `ldap` database type (which is the standard user database).

To use your own methods and database types, you must register them.

Attributes, Methods, and Database Types

Attributes, methods, and database types are interrelated. When you register an attribute getter function (see "Writing the Initialization Function"), you associate the getter function with a method and a database type.

For each attribute that you define, you can register a number of different attribute getter functions. When you call the `ACL_GetAttribute()` function to get the value of an attribute, the function calls each attribute getter function until one of them does not return `LAS_EVAL_DECLINE`.

An attribute getter function is matching if the current method and database type match the method and database type associated with the attribute getter function.

Also, the method type `ACL_METHOD_ANY` matches any method, and the database type `ACL_DBTYPE_ANY` matches any database type. If you register an attribute getter function with `ACL_METHOD_ANY` and `ACL_DBTYPE_ANY`, the attribute getter function will always match.

Registering Methods

To register a method, call the `ACL_MethodRegister()` function. You can also register a method by adding the following line to your `obj.conf` file:

```
Init fn=acl-register-method method="nameOfYourMethod"
```

Do not insert any line breaks or carriage returns within each statement.

Registering Database Types

A *database type* represents information about a specific type of database. Each database type is associated with a parsing function. This parsing function is called when you register a database of this type. The parsing function parses a reference to a database (which is passed in as an argument when the database is registered) and extracts information about the database.

For example, the standard database type that is registered by the server during startup is `ldap`.

To register a database type, call the ACL_DbTypeRegister() function.

Registering Databases

When you register an authentication database, you need to specify the database type and the URL to the database. The URL to the database is parsed by the parsing function associated with the database type (see "Registering Database Types"). The parsed data is kept as part of the registered information (you can retrieve it by calling the ACL_DatabaseFind() function).

To register a database, call the ACL_DatabaseRegister() function. You can also register a database by adding the following line to your obj.conf file:

```
Init fn=acl-register-dbname dbtype="typeOfYourDatabase"
dbname="nameOfYourDatabase" url="urlToYourDatabase"
```

Note that you should not insert any line breaks or carriage returns within each statement.

In addition to calling the ACL_DatabaseRegister() function and adding lines to the obj.conf file, you can register a database by adding entries to the dbswitch.conf file.

Setting the Default Method and Database

You can set the default method and database type used either by calling API functions or by setting up directives in your obj.conf file.

Setting the Default Method

To specify the default method, call the ACL_MethodSetDefault() function. You can also set the default method by adding the following line to your obj.conf file:

```
Init fn=acl-set-default-method method="nameOfYourMethod"
```

Note that you should not insert any line breaks or carriage returns within each statement.

Setting the Default Database

To specify the default database, call the `ACL_DatabaseSetDefault()` function. You can also set the default database by adding the following line to your `obj.conf` file:

```
Init fn=acl-set-default-database dbname="nameOfYourDatabase"
```

Note that you should not insert any line breaks or carriage returns within each statement.

Setting Up New Database Types

There are several different reasons to set up your own non-LDAP database. For example, you already have information in a database and it is not feasible to migrate it to LDAP. You might also be using applications which only work with your database type.

Setting up a new database type involves more than just registering your new database. You also need to register the attribute getter function and the parsing function.

An example of source code for setting up a custom database is located on-line at `http://developer.netscape.com/library/examples/nsapi/customdb.html`.

Chapter

7

Examples of LAS Modules

This chapter contains examples that demonstrate how you can use the Access Control API to write your own LAS modules.

- **Email addresses used for authentication.** The first example shows how you can authenticate users based on their email addresses.

  For information on defining an LAS module that does this, see "LAS Code for the Email Attribute". This example is also included in the `plugins/nsacl` (or `plugins\nsacl`) directory under your server root directory.

- **Referer URLs used for authentication.** This example shows how to use the HTTP referer data from HTTP headers to determine the access rights of a user visiting the site.

  For information on defining an LAS module that does this, see "LAS Code for the HTTP Referer Attribute". This example is also included in the `plugins/nsacl` (or `plugins\nsacl`) directory under your server root directory.

LAS Code for the Email Attribute

Suppose the user is attempting to access a resource that has the following ACL:

```
version 3.0.1;
acl email;
authenticate (email) {
```

```
   method=”basic”;
   database=”any”
};
deny (all)
   email= “anyone”;
allow (read)
   email = “mozilla@netscape.com”;
```

Authentication in this ACL is based on the user's email address, so the server needs to be able to get the email address and determine if it is equal to mozilla@netscape.com.

The source file for this example (`lasemail.c`) contains functions that do this. The functions define an LAS for the email attribute:

- `LASEmailModuleInit()`

  This function registers the LAS for the email attribute. As part of the registration process, this function identifies the functions that:

  - Get and evaluate the value of the email attribute
  - Flush any authentication information that is cached

  For details on how this function is defined, see “Registering the LAS for the Email Attribute”.

- `LASEmailEval()`

  This function gets the email address of the user and compares it against the values in the access control entries.

  To get the email address, this function gets the name of the user and searches the directory for the user's entry.

  For details on how this function is defined, see “Evaluating the Email Attribute Expression”.

- `LASEmailFlush()`

  This function flushes any cached authentication data.

  Because the process of evaluating the attribute value does not produce data that is cached between evaluations, the function for flushing the cache is an empty function that does nothing.

For details on how this function is defined, see “Flushing the Cache for the Email Attribute”.

The custom attribute “email” is globally defined as `ACL_ATTR_EMAIL`.

Evaluating the Email Attribute Expression

The `LASEmailEval()` function evaluates the attribute expression for the email attribute. The definition of this function is shown below.

```
int LASEmailEval(NSErr_t *errp, char *attr_name, CmpOp_t comparator,
       char *attr_pattern, ACLCachable_t *cachable,
       void **LAS_cookie, PList_t subject, PList_t resource,
       PList_t auth_info, PList_t global_auth)
{
   char *uid;
   char *email;
   int rv;
   LDAP *ld;
   char *basedn;
   LDAPMessage *res;
   int numEntries;
   char filter[1024];
   int matched;

   /* Results of the evaluation will not be cached. */
   *cachable = ACL_NOT_CACHABLE;
   *LAS_cookie = (void *)0;

   /* Verify that this function is only called when the email
   attribute expression is being evaluated. */

   if (strcasecmp(attr_name, ACL_ATTR_EMAIL) != 0) {
      fprintf(stderr, “LASEmailEval called for incorrect attr \”%s\”\n”,
         attr_name);
      return LAS_EVAL_INVALID;
   }

   /* The email attribute supports only “equals” and “not equals”. */

   if ((comparator != CMP_OP_EQ) && (comparator != CMP_OP_NE)) {
      fprintf(stderr, “LASEmailEval called with wrong comparator %d\n”,
         comparator);
      return LAS_EVAL_INVALID;
   }
```

```
if (!strcmp(attr_pattern, “anyone”)) {
   return comparator == CMP_OP_EQ ? LAS_EVAL_TRUE : LAS_EVAL_FALSE;
}

/* Get the authenticated user name. */

rv = ACL_GetAttribute(errp, ACL_ATTR_USER, (void **)&uid,
   subject, resource, auth_info, global_auth);

if (rv != LAS_EVAL_TRUE) {
   return rv;
}

/* The user is authenticated. */

if (!strcmp(attr_pattern, “all”)) {
   return comparator == CMP_OP_EQ ? LAS_EVAL_TRUE : LAS_EVAL_FALSE;
}

/* Get the handle of the directory database. */

rv = ACL_LDAPDatabaseHandle(errp, NULL, &ld, &basedn);

if (rv != LAS_EVAL_TRUE) {
   fprintf(stderr, “unable to get LDAP handle\n”);
   return rv;
}

/* Formulate the filter -- assume single e-mail in attr_pattern. */

/* If we support multiple comma separated e-mail addresses in the
 * attr_pattern then the filter will look like:
 *     (& (uid=<user>) (| (mail=<email1>) (mail=<email2>)))
 */

sprintf(filter, “(& (uid=%s) (mail=%s))”, uid, attr_pattern);

/* Perform the search. */

rv = ldap_search_s(ld, basedn, LDAP_SCOPE_SUBTREE, filter,
   0, 0, &res);

if (rv != LDAP_SUCCESS) {
   fprintf(stderr, "ldap_search_s: %s\n", ldap_err2string(rv));
   return LAS_EVAL_FAIL;
}

numEntries = ldap_count_entries(ld, res);

if (numEntries == 1) {
   /* success */
   LDAPMessage *entry = ldap_first_entry(ld, res);
   char *dn = ldap_get_dn(ld, entry);
   fprintf(stderr, "ldap_search_s: Entry found. DN: \"%s\"\n", dn);
   ldap_memfree(dn);
   matched = 1;
```

```
   } else if (numEntries == 0) {
      /* not found -- but not an error */
      fprintf(stderr, "ldap_search_s: Entry not found.
         Filter: \"%s\"\n", filter);
      matched = 0;

   } else if (numEntries > 0) {
      /* Found more than one entry, so return an error. */
      fprintf(stderr, "ldap_search_s: Found more than one entry.
         Filter: \"%s\ "\n", filter);
      return LAS_EVAL_FAIL;
   }

   /* Return LAS_EVAL_TRUE if attribute value matches the expression. */

   if (comparator == CMP_OP_EQ) {
      rv = (matched ? LAS_EVAL_TRUE : LAS_EVAL_FALSE);
   } else {
      rv = (matched ? LAS_EVAL_FALSE : LAS_EVAL_TRUE);
   }

   return rv;
}
```

Flushing the Cache for the Email Attribute

Because the results of the attribute evaluation are not cached, the `LASEmailFlush()` function does not do anything. The definition of this function is shown below.

```
void LASEmailFlush(void **las_cookie)
{
   /* do nothing */
   return;
}
```

Registering the LAS for the Email Attribute

The `LASEmailModuleInit()` function registers the LAS for the email attribute. The definition of this function is shown below.

```
int LASEmailModuleInit ()
{
   NSErr_t err = NSERRINIT;
   int rv;

   /* Register the LAS. Associate the attribute with the functions
   used to evaluate attribute expressions and flush cached results. */

   rv = ACL_LasRegister(&err, ACL_ATTR_EMAIL, LASEmailEval,
      LASEmailFlush);

   if (rv < 0) {
      fprintf(stderr, "ACL_LasRegister failed.  Error: %d\n", rv);
      return rv;
   }
}
```

To load this function in the server, compile the file into a shared object or dynamic link library (`lasemail.so` or `lasemail.dll`) and add the following lines to the `obj.conf` file:

```
Init fn="load-modules" funcs="LASEmailModuleInit" shlib="lasemail.so"

Init fn=acl-register-module module="lasemail" func="LASEmailModuleInit"
```

Make sure to replace the name of the library (`lasemail.so`) with the full pathname to the library (for example, `/nshome/lib/lasemail.so`).

To register the acl on the server, add the following line to the `obj.conf` file:

```
PathCheck fn="check-acl" acl="email"
```

Note You only need to register the acl on the server if you haven't associated any ACLs with your resource.

LAS Code for the HTTP Referer Attribute

Suppose you want to control access based on the value of the HTTP referer attribute. The source file for the example (`lasref.c`) contains functions that do this. The following is an example of how an ACL for the HTTP referer would look:

```
version 3.0.1;
acl referer;
authenticate (referer) {
   method=”basic”;
   database=”any”
};
deny (all)
   referer= “anyone”;
allow (read)
   referer = “http://*.netscape.com”;
```

The functions define an LAS for the referer attribute:

- `las_ref_init()`

  This function registers the LAS for the referer attribute. As part of the registration process, this function identifies the functions that:

  - Get the referer attribute
  - Evaluate expressions with the referer attribute
  - Flush any authentication information that is cached

  For details on how this function is defined, see “Registering the LAS for the Referer Attribute”.

- `las_ref_get()`

  This function gets the referer attribute.

  For details on how this function is defined, see “Getting the Value of the Referer Attribute”.

- `las_ref_eval()`

  This function compares the referer attribute against the values in the access control entries.

For details on how this function is defined, see “Evaluating the Referer Attribute Expression”.

- `las_ref_flush()`

  This function flushes any cached authentication data.

  Because the process of evaluating the attribute value does not produce data that is cached between evaluations, the function for flushing the cache is an empty function that does nothing.

  For details on how this function is defined, see “Flushing the Cache for the Referer Attribute”.

The custom attribute “referer” is globally defined as `LASREF_ATTRNAME`.

Getting the Value of the Referer Attribute

The `las_ref_get()` function gets the referer from the request HTTP headers and adds it to the subject property list. This type of function is sometimes called a “getter” function.

For a given request, this function is called the first time the referer attribute is encountered in the ACL, typically when an attribute expression evaluator calls the `ACL_GetAttribute()` function.

```
int las_ref_get(NSErr_t *errp, PList_t subject, PList_t resource,
   PList_t auth_info, PList_t global_auth, void *arg)
{
   Request *rq;
   char *referer;
   int rv;

   /* Get pointer to NSAPI Req structure from resource property list */

   rv = PListGetValue(resource, ACL_ATTR_REQUEST_INDEX, (void **)&rq,
      NULL);

   if (rv < 0) {
      ereport(LOG_FAILURE, “las-ref-get missing request”);
      return LAS_EVAL_FAIL;
   }

   /* Try to get the HTTP referer header */
```

```
   referer = pblock_findval("referer", rq->headers);

   if (referer) {
      /* Log referer string for debug purposes */
      ereport(LOG_INFORM, "las-ref-get: referer=%s", referer);

      /* Add the referer to the subject property list */
      rv = PListInitProp(subject, 0, LASREF_PROPNAME, STRDUP(referer),
         NULL);

      if (rv < 0) {
         ereport(LOG_FAILURE,
            "las-ref-get cannot add request to subject property list");
         return LAS_EVAL_FAIL;
      }

      /* Successfully retrieved referer */
      return LAS_EVAL_TRUE;
   }

   /* Referer not found */
   return LAS_EVAL_FAIL;
}
```

Evaluating the Referer Attribute Expression

The `las_ref_eval()` function evaluates the attribute expression for the referer attribute. It calls the `ACL_GetAttribute()` function to get the value of the referer attribute, and then evaluates it against a comparison operator and pattern string. Comparisons are case-insensitive.

The following table summarizes the comparison operators.

Table 7.1 Comparison operators and pattern strings

Example	Description
referer = "pattern"	The referer must match the pattern.
referer != "pattern"	The referer must not match the pattern.
referer < "pattern" referer <= "pattern"	The referer must be a prefix of the pattern.
referer > "pattern" referer >= "pattern"	The pattern must be a prefix of the referer.

The definition of this function is shown below.

```
int las_ref_eval(NSErr_t *errp, char *attr_name, CmpOp_t comparator,
   char *attr_pattern, ACLCachable_t *cachable, void **cookie,
   PList_t subject, PList_t resource, PList_t auth_info,
   PList_t global_auth)
{
   char *referer;
   int patlen;
   int reflen;
   int rv;

   /* Get the value of the "referer" attribute */

   rv = ACL_GetAttribute(errp, LASREF_ATTRNAME, (void **)&referer,
      subject, resource, auth_info, global_auth);

   if (rv == LAS_EVAL_TRUE) {

      /* If there is a pattern string, evaluate the comparator */

      if (attr_pattern) {

         switch (comparator) {
         case CMP_OP_EQ:

            /* Case-insensitive check for equality */
            rv = strcasecmp(referer, attr_pattern) ? LAS_EVAL_FALSE
               : LAS_EVAL_TRUE;
            break;

         case CMP_OP_NE:

            /* Case-insensitive check for inequality */
            rv = strcasecmp(referer, attr_pattern) ? LAS_EVAL_TRUE
               : LAS_EVAL_FALSE;
            break;

         case CMP_OP_LT:
         case CMP_OP_LE:

            /* Case-insensitive check if referer is prefix of pattern */
            reflen = strlen(referer);
            patlen = strlen(attr_pattern);

            rv = ((reflen > patlen) ||
               strncasecmp(referer, attr_pattern, reflen))
               ? LAS_EVAL_FALSE : LAS_EVAL_TRUE;
            break;

         case CMP_OP_GT:
         case CMP_OP_GE:

            /* Case-insensitive check if pattern is prefix of referer */
            reflen = strlen(referer);
            patlen = strlen(attr_pattern);
```

```
            rv = ((patlen > reflen) ||
               strncasecmp(referer, attr_pattern, patlen))
               ? LAS_EVAL_FALSE : LAS_EVAL_TRUE;
            break;
         }
      }

      return rv;
   }

   return LAS_EVAL_FAIL;
}
```

Flushing the Cache for the Referer Attribute

Because the results of the attribute evaluation are not cached, the `las_ref_flush()` function does not do anything. The definition of this function is shown below.

```
void las_ref_flush(void **cookie)
{
   /* Nothing to do */
   return;
}
```

Registering the LAS for the Referer Attribute

The `las_ref_init()` function registers the LAS for the referer attribute. The definition of this function is shown below.

```
NSAPI_PUBLIC int las_ref_init(NSErr_t *errp)
{
   int rv;

   /* Register the function for getting the referer attribute. */

   rv = ACL_AttrGetterRegister(errp, LASREF_ATTRNAME, las_ref_get,
         ACL_METHOD_ANY, ACL_DBTYPE_ANY, ACL_AT_FRONT, NULL);

   if (rv < 0) {
      fprintf(stderr,
         "las_ref_init: ACL_AttrGetterRegister error: %d\n", rv);
```

```
    return rv;
  }

  /* Register the LAS. */

  rv = ACL_LasRegister(errp, LASREF_ATTRNAME, las_ref_eval,
        las_ref_flush);

  if (rv < 0) {
    fprintf(stderr,
      "las_ref_init: ACL_LasRegister error: %d\n", rv);
    return rv;
  }

  return 0;
}
```

To load this function in the server, compile the file into a shared object or dynamic link library (`lasref.so` or `lasref.dll`) and add the following lines to the `obj.conf` file:

```
Init fn="load-modules" funcs="las-ref-init" shlib=server-root/plugins/
nsacl/lasref.so"

Init fn="acl-register-module" module="lasref" func="las-ref-init"
```

Make sure to replace the name of the library (`lasref.so`) with the full pathname to the library (for example, `/nshome/lib/lasref.so`).

To register the acl on the server, add the following line to the `obj.conf` file:

```
PathCheck fn="check-acl" acl="referer"
```

Note You only need to register the acl on the server if you haven't associated any ACLs with your resource.

Chapter

8

Data Types and Structures

This chapter describes the data structures and data structure types used by functions in the Access Control API. They are listed alphabetically.

ACLDbType_t

`ACLDbType_t` represents a reference to a type of authentication database.

When you register a new type of authentication database by calling the `ACL_DbTypeRegister()` function, a new reference is returned.

You can get the reference to a registered database type from the type name by calling the `ACL_DbTypeFind()` function.

See Also `ACL_DbTypeRegister()`, `ACL_DbTypeIsEqual()`, `ACL_DbTypeNameIsEqual()`, `ACL_DbTypeGetDefault()`, `ACL_DbTypeFind()`, `ACL_AuthInfoGetDbType()`, `ACL_AttrGetterRegister()`, `ACL_DatabaseRegister()`, `ACL_DatabaseFind()`.

ACLEvalHandle_t

`ACLEvalHandle_t` represents an ACL evaluation context, which is used to determine if a given subject has certain access rights to a given resource. The evaluation context includes the ACL list and the property lists for the subject and resource.

See Also `ACL_EvalNew()`, `ACL_EvalDestroy()`, `ACL_EvalSetACL()`, `ACL_EvalGetResource()`, `ACL_EvalSetResource()`, `ACL_EvalGetSubject()`, `ACL_EvalSetSubject()`, `ACL_EvalTestRights()`.

ACLEvalRes_t

`ACLEvalRes_t` represents the result of an ACL evaluation. The result can be one of the following:

Table 8.1 Results of evaluating an ACL

Result	Description
`ACL_RES_ALLOW`	Subject is allowed access to the resource.
`ACL_RES_DENY`	Subject is denied access to the resource.
`ACL_RES_FAIL`	The ACL could not be evaluated (for example, an error occurred).

Table 8.1 Results of evaluating an ACL (Continued)

Result	Description
ACL_RES_INVALID	The ACL evaluation context is not valid (for example, the comparison operator in the ACL is not valid for this attribute).
ACL_RES_NONE	Currently unused.

ACLExprHandle_t

ACLExprHandle_t represents an in-memory access control entry from an ACL. An ACLExprHandle_t can represent an allow statement, a deny statement, an authenticate statement, or a deny with URI statement.

See Also ACL_ExprNew(), ACL_ExprDestroy(), ACL_ExprAddArg(), ACL_ExprTerm(), ACL_ExprAnd(), ACL_ExprOr(), ACL_ExprNot(), ACL_ExprAddAuthInfo(), ACL_ExprGetDenyWith(), ACL_ExprSetDenyWith(), ACL_ExprAppend(), ACL_ExprSetPFlags(), ACL_ExprClearPFlags().

ACLExprType_t

ACLExprType_t represents the type of an access control entry in an ACL. Different types of statements in an ACL file correspond to different types of access control entries.

When you call the ACL_ExprNew() function to create a new access control entry, you need to specify one of the following types:

Table 8.2 Types of access control entries

Bit Flag	Equivalent Type of Statement in an ACL File	Description
ACL_EXPR_TYPE_ALLOW	allow	Allows access to a resource.
ACL_EXPR_TYPE_DENY	deny	Denies access to a resource.

Table 8.2 Types of access control entries (Continued)

Bit Flag	Equivalent Type of Statement in an ACL File	Description
ACL_EXPR_TYPE_AUTH	authenticate	Specifies the method of authentication and the authentication database to use.
ACL_EXPR_TYPE_RESPONSE	deny with	Specifies the URL to which the user is directed if the user is denied access to the resource.

See Also ACL_ExprNew().

ACLHandle_t

ACLHandle_t represents an ACL in memory.

To get an ACLHandle_t from an ACLListHandle_t, call the ACL_ListFind() function. To get and manipulate ACLs directly from ACL files, call the ACL_FileGetAcl(), ACL_FileSetAcl(), ACL_FileRenameAcl(), and ACL_FileDeleteAcl() functions.

See Also ACL_AclNew(), ACL_AclDestroy(), ACL_ListFind(), ACL_ListAclDelete(), ACL_FileGetAcl(), ACL_FileSetAcl(), ACL_FileRenameAcl(), ACL_FileDeleteAcl().

ACLListEnum_t

ACLListEnum_t contains the current position of an enumeration in an ACLListHandle_t structure.

To get an ACLHandle_t from an ACLListHandle_t, call the ACL_ListFind() function. To get and manipulate ACLs directly from ACL files, call the ACL_FileGetAcl(), ACL_FileSetAcl(), ACL_FileRenameAcl(), ACL_FileDeleteAcl() functions.

See Also ACL_ListGetFirst(), ACL_ListGetNext().

ACLListHandle_t

`ACLListHandle_t` represents a list of ACLs in memory.

You can read a list of ACLs into this in-memory form by calling the `ACL_ParseFile()` and `ACL_ParseString()` functions.

To save the in-memory ACLs to an ACL file, call the `ACL_WriteFile()` function, or to save to a string buffer, call the `ACL_WriteString()` function.

See Also `ACL_ParseFile()`, `ACL_ParseString()`, `ACL_WriteFile()`, `ACL_WriteString()`, `ACL_ListAppend()`, `ACL_ListNew()`, `ACL_ListDestroy()`, `ACL_ListGetNameList()`, `ACL_NameListDestroy()`, `ACL_ListAclDelete()`, `ACL_ListConcat()`, `ACL_ListFind()`.

ACLMethod_t

`ACLMethod_t` represents a reference to an authentication method.

When you register a new authentication method by calling the `ACL_MethodRegister()` function, a new reference is returned.

You can get the reference to a registered method from the method's name by calling the `ACL_MethodFind()` function.

See Also `ACL_MethodRegister()`, `ACL_MethodIsEqual()`, `ACL_MethodNameIsEqual()`, `ACL_MethodGetDefault()`, `ACL_MethodSetDefault()`, `ACL_MethodFind()`, `ACL_AuthInfoGetMethod()`, `ACL_AttrGetterRegister()`.

CmpOp_t

`CmpOp_t` represents the comparison operators used in an access control entry in an ACL (for example, `user = mozilla`).

You can call the `ACL_ExprTerm()` function to define a term that uses one of the comparison operators:

- `CMP_OP_EQ`

Equality operator (=)

- `CMP_OP_NE`

  Not Equals operator (=)

- `CMP_OP_GT`

  Greater Than operator (=)

- `CMP_OP_LT`

  Less Than operator (=)

- `CMP_OP_GE`

  Greater Than Or Equal To operator (=)

- `CMP_OP_LE`

  Less Than Or Equal To operator (=)

See Also `ACL_ExprTerm()`.

NSEFrame_t

`NSEFrame_t` represents the structure of an error frame, defined as follows:

```
typedef struct NSEFrame_s NSEFrame_t;

struct NSEFrame_s {
   NSEFrame_t * ef_next
   long ef_retcode;
   long ef_errorid;
   char * ef_program;
   int ef_errc;
   char * ef_errv[NSERRMAXARG];
};
```

The fields in this structure are described below:

`ef_next`	Pointer to the next error frame in the `NSErr_t` list.
`ef_retcode`	Error return code identifying the type of error that occurred.
`ef_errorid`	Error ID which uniquely identifies errors in a module or library.

`ef_program`	String value that identifies the library or module context for the `er_erroid` argument.
`ef_errc`	Number of strings in the `ef_errv[]` array for the current error ID.
`ef_errv`	Array of string values relevant to the current error ID. You can include these strings in an error message retrieved from a message file. The strings in a message file can contain "`%s`" `sprintf()` format codes. The `ef_errv[]` strings are passed to `sprintf()` along with the error message string.

NSErr_t

`NSErr_t` represents the structure of a header for a list of error frames.

The header contains a pointer to the first and last error frames on the list. The first error frame is normally the one most recently generated, which usually represents the highest-level interpretation available for an error that is propagating upwards in a call chain.

These structures are generally allocated as automatic or static variables.

`NSErr_t` is defined as follows:

```
typedef struct NSErr_s NSErr_t;

struct NSErr_s {
   NSEFrame_t * err_first;
   NSEFrame_t * err_last;
   NSEFrame_t *(*err_falloc)(NSErr_t * errp);
   void (*err_ffree)(NSErr_t * errp, NSEFrame_t * efp);
};
```

The fields in this structure are described below:

`err_first`	Pointer to the first error frame in the list.
`err_last`	Pointer to the last error frame in the list.
`err_falloc`	Pointer to a function for allocating memory for error frames.
`err_ffree`	Pointer to a function for freeing error frames from memory.

PFlags_t

`PFlags_t` represents the processing flags that apply to an access control entry in an ACL.

You can call the `ACL_ExprSetPFlags()` function to set the following bit flags for an access control entry:

Table 8.3 Bit flags that you can apply to access control entries

Bit Flag	Equivalent Keyword in ACL File	Description
`ACL_PFLAG_ABSOLUTE`	`absolute`	Specifies that the results of the current expression override the results of any subsequent expressions. For example, if a directory's ACL denies access and has this flag set, any resources within the directory (or its subdirectories) cannot be accessed, even if those resources have ACLs that allow access.

Table 8.3 Bit flags that you can apply to access control entries (Continued)

Bit Flag	Equivalent Keyword in ACL File	Description
ACL_PFLAG_TERMINAL	static	Specifies that the ACL only applies to its associated container object itself, not to resources or container objects within this object. For example, if a directory's ACL has this flag set, this ACL is not considered when evaluating access to resources or subdirectories within the directory.
ACL_PFLAG_CONTENT	content	Specifies that the ACL applies to the contents of its associated container object, not to the container itself. For example, if a directory's ACL has this flag set, the ACL applies only to the resources and subdirectories within the directory. The ACL does not apply to the directory itself.

See Also ACL_ExprSetPFlags(), ACL_ExprClearPFlags().

Chapter

9

Functions and Function Typedefs

This chapter describes the public functions of the Access Control API. Each description includes the name of the function, its header file, its syntax, what it returns, its parameters, and an example of its use.

This chapter organizes functions as follows:

- By task
- Alphabetically

Summary of Functions by Task

This section summarizes the functions that you can call to perform these tasks:

- Parsing ACLs
- Manipulating ACLs in Files
- Manipulating ACLs in Memory
- Constructing ACL Expressions
- Evaluating ACLs
- Working with ACL Lists

- Defining a Loadable Authentication Service (LAS)
- Getting Attribute Values
- Working with Authentication Methods and Databases
- Working with Property Lists
- Working with Error Frames
- Multithreading

Parsing ACLs

Call the following functions to parse ACL strings and read them into memory.

Table 9.1 Functions for parsing ACLs

To do this:	Call this function:
Read an ACL list from an ACL file into memory.	`ACL_ParseFile()`
Read a string of text definitions of ACLs into memory as an ACL list.	`ACL_ParseString()`

Manipulating ACLs in Files

Call the following functions to read, modify, and write ACL files.

Table 9.2 Functions for working with ACL files

To do this:	Call this function:
Read an ACL list from an ACL file.	`ACL_ParseFile()`
Save an ACL list to an ACL file.	`ACL_WriteFile()`
Get an ACL from an ACL file.	`ACL_FileGetAcl()`
Set an ACL in an ACL file.	`ACL_FileSetAcl()`
Rename an ACL in an ACL file.	`ACL_FileRenameAcl()`
Remove an ACL from an ACL file.	`ACL_FileDeleteAcl()`

Manipulating ACLs in Memory

Call the following functions to create, modify, or destroy ACLs.

Table 9.3 Functions for working with ACLs

To do this:	Call this function:
Create a new ACL.	ACL_AclNew()
Free an ACL from memory.	ACL_AclDestroy()
Get the name from an ACL.	ACL_AclGetTag()
Find an ACL by name in an ACL list.	ACL_ListFind()
Remove an ACL from an ACL list.	ACL_ListAclDelete()
Flush the ACL cache.	ACL_CacheFlush()

Constructing ACL Expressions

Call the following functions to add, modify, and destroy expressions in ACLs. These functions are part of a low-level interface that may be useful for defining ACLs without using the ACL file syntax. However, future support of this low-level interface is not guaranteed. Wherever possible, use the ACL_ParseFile() and ACL_ParseString() functions instead.

Table 9.4 Functions for working with expressions in ACLs

To do this:	Call this function:
Create a new access control expression.	ACL_ExprNew()
Free an expression from memory.	ACL_ExprDestroy()
Add a term (for example, user = "mozilla") to an expression.	ACL_ExprTerm()
Combine two terms with a boolean AND operator (for example, user = "mozilla" and dns != "*.netscape.com").	ACL_ExprAnd()
Combine two terms with a boolean OR operator (for example, user="mozilla" or dns !="*.netscape.com).	ACL_ExprOr()
Precede a term with a boolean NOT operator (for example, not group = "administrators").	ACL_ExprNot()

Table 9.4 Functions for working with expressions in ACLs (Continued)

To do this:	Call this function:
Add an argument to an expression. Call this function to add rights (such as read, write, or execute) to an allow or deny statement. You can also call this function to add attributes (such as user or group) to an authenticate statement.	ACL_ExprAddArg()
Add a name-value pair to the authentication information (for example, method="basic").	ACL_ExprAddAuthInfo()
Append an expression to an ACL.	ACL_ExprAppend()
Get the URL or URI used in a "deny with" expression (such as "deny with url=http://www.myhost.com/error.html")	ACL_ExprGetDenyWith()
Specify the URL or URI used in a "deny with" expression. This is the URL returned to a user who is denied access to a resource.	ACL_ExprSetDenyWith()
Set the processing flags associated with an expression in an ACL.	ACL_ExprSetPFlags()
Clear the processing flags associated with an expression in an ACL.	ACL_ExprClearPFlags()

Evaluating ACLs

Call the following functions to determine whether a subject has access rights to a resource.

Table 9.5 Functions for evaluating ACLs

To do this:	Call this function:
Create a new evaluation context.	ACL_EvalNew()
Free an evaluation context from memory.	ACL_EvalDestroy()
Set the ACL list associated with an evaluation context.	ACL_EvalSetACL()
Set the resource property list associated with an evaluation context.	ACL_EvalSetResource()
Set the subject property list associated with an evaluation context.	ACL_EvalSetSubject()

Table 9.5 Functions for evaluating ACLs (Continued)

To do this:	Call this function:
Get the resource property list associated with an evaluation context.	ACL_EvalGetResource()
Get the subject property list associated with an evaluation context.	ACL_EvalGetSubject()
Determine if the specified subject has access rights to the specified resource.	ACL_EvalTestRights()
Returns the default result of an ACL evaluation.	ACL_GetDefaultResult()
Determine if the subject has rights to access a resource.	ACL_SetupEval()
Set the default result of an ACL evaluation.	ACL_SetDefaultResult()

Working with ACL Lists

Call the following functions to create, modify, or destroy ACL lists.

Table 9.6 Functions for working with ACL lists

To do this:	Call this function:
Create a new ACL list in memory (as an ACLListHandle_t).	ACL_ListNew()
Free an ACL list (ACLListHandle_t) from memory.	ACL_ListDestroy()
Find an ACL by name in an ACL list.	ACL_ListFind()
Iterate through the ACLs in an ACL list.	ACL_ListGetFirst(), ACL_ListGetNext()
Remove an ACL from an ACL list.	ACL_ListAclDelete()
Append an ACL to an ACL list.	ACL_ListAppend()
Concatenate two ACLs together.	ACL_ListConcat()
Get a list of the names of the ACLs in an ACL list.	ACL_ListGetNameList()
Free a list of ACL names from memory.	ACL_NameListDestroy()

Table 9.6 Functions for working with ACL lists (Continued)

To do this:	Call this function:
Read an ACL list from an ACL file or from a string.	ACL_ParseFile(), ACL_ParseString()
Write an ACL list to an ACL file or to a string.	ACL_WriteFile(), ACL_WriteString()

Defining a Loadable Authentication Service (LAS)

Call the following functions to define your own Loadable Authentication Service (LAS).

Table 9.7 Functions for defining loadable authentication services

To do this:	Call this function:
Register an LAS.	ACL_LasRegister()
Define a function for evaluating attribute expressions.	LASEvalFunc_t
Define a function for flushing attribute data from the cache.	LASFlushFunc_t

Getting Attribute Values

Call the following functions to get the value of an attribute.

Table 9.8 Functions for getting attribute values

To do this:	Call this function:
Register a function used to get the value of an attribute.	ACL_AttrGetterRegister()
Define a function that gets the value of an attribute.	ACLAttrGetterFn_t
Get the value of an attribute.	ACL_GetAttribute()

Working with Authentication Methods and Databases

Call the following functions when working with authentication methods and databases.

Table 9.9 Functions for working with authentication methods and databases

To do this:	Call this function:
Register an LAS module.	ACL_ModuleRegister(), AclModuleInitFunc
Register a database type and its corresponding parsing function.	ACL_DbTypeRegister(), DbParseFn_t
Get the type of the database specified in authentication information.	ACL_AuthInfoGetDbType()
Find a registered database type.	ACL_DbTypeFind()
Get the default type of database.	ACL_DbTypeGetDefault()
Determine if a given database type is registered.	ACL_DbTypeIsRegistered()
Determine if two given database types are equal.	ACL_DbTypeIsEqual()
Determine if a given database with a name uses a specified database type.	ACL_DbTypeNameIsEqual()
Get the parsing function associated with a database type.	ACL_DbTypeParseFn()
Register an authentication database.	ACL_DatabaseRegister()
Get the name of the database specified in authentication information.	ACL_AuthInfoGetDbname()
Find a registered authentication database.	ACL_DatabaseFind(), ACL_LDAPDatabaseHandle()
Change the default database and the default type of database used for authentication.	ACL_DatabaseSetDefault()
Returns the name of the default database.	ACL_DatabaseGetDefault
Register an authentication method.	ACL_MethodRegister()
Get the authentication method specified in authentication information.	ACL_AuthInfoGetMethod()
Find a registered authentication method.	ACL_MethodFind()

Table 9.9 Functions for working with authentication methods and databases (Continued)

To do this:	Call this function:
Get the default authentication method.	ACL_MethodGetDefault()
Set the default authentication method.	ACL_MethodSetDefault()
Determine if two authentication methods are equal.	ACL_MethodIsEqual()
Determine if a method with a given name is the specified type of method.	ACL_MethodNameIsEqual()

Working with Property Lists

Call the following functions to create, modify, or destroy property lists.

Table 9.10 Functions for working with property lists

To do this:	Call this function:
Create a new property list.	PListCreate(), PListNew()
Free a property list from memory.	PListDestroy()
Copy a property list.	PListDuplicate()
Add a new property to a property list.	PListDefProp(), PListInitProp()
Remove a property from a property list.	PListDeleteProp()
Iterate through each property in a property list	PListEnumerate(), PListFunc_t
Get the memory pool used by a property list.	PListGetPool()
Assign a name to a property.	PListNameProp()
Set the value and type of a property.	PListAssignValue(), PListSetValue(), PListSetType()
Get the value and type of a property.	PListFindValue(), PListGetValue()

Working with Error Frames

Call the following functions to create, modify, or destroy error frames.

Table 9.11 Functions for working with error frames

To do this:	Call this function:
Allocate memory for a new error frame structure.	`nserrFAlloc()`
Free an error frame structure from memory.	`nserrFFree()`
Add a new error frame to a list of error frames.	`nserrGenerate()`
Free a list of error frames from memory.	`nserrDispose()`

Multithreading

Call the following functions to create critical regions when required for multithreading.

Table 9.12 Functions for multithreading

To do this:	Call this function:
Start a critical region, blocking other threads.	`ACL_CritEnter()`
Finish a critical region, unblocking other threads.	`ACL_CritExit()`

Functions and Function Type Definitions

This section describes the functions and function type definitions in the Access Control API. They are listed alphabetically.

- ACLAttrGetterFn_t
- AclModuleInitFunc
- ACL_AclDestroy()
- ACL_AclGetTag()
- ACL_AclNew()

- ACL_AttrGetterRegister()
- ACL_AuthInfoGetDbname()
- ACL_AuthInfoGetDbType()
- ACL_AuthInfoGetMethod()
- ACL_CacheFlush()
- ACL_CritEnter()
- ACL_CritExit()
- ACL_DatabaseFind()
- ACL_DatabaseGetDefault()
- ACL_DatabaseRegister()
- ACL_DatabaseSetDefault()
- ACL_DbTypeFind()
- ACL_DbTypeGetDefault()
- ACL_DbTypeIsEqual()
- ACL_DbTypeIsRegistered()
- ACL_DbTypeNameIsEqual()
- ACL_DbTypeParseFn()
- ACL_DbTypeRegister()
- ACL_EvalDestroy()
- ACL_EvalGetResource()
- ACL_EvalGetSubject()
- ACL_EvalNew()
- ACL_EvalSetACL()
- ACL_EvalSetResource()

- ACL_EvalSetSubject()
- ACL_EvalTestRights()
- ACL_ExprAddArg()
- ACL_ExprAddAuthInfo()
- ACL_ExprAnd()
- ACL_ExprAppend()
- ACL_ExprClearPFlags()
- ACL_ExprDestroy()
- ACL_ExprGetDenyWith()
- ACL_ExprNew()
- ACL_ExprNot()
- ACL_ExprOr()
- ACL_ExprSetDenyWith()
- ACL_ExprSetPFlags()
- ACL_ExprTerm()
- ACL_FileDeleteAcl()
- ACL_FileGetAcl()
- ACL_FileRenameAcl()
- ACL_FileSetAcl()
- ACL_GetAttribute()
- ACL_GetDefaultResult()
- ACL_LasRegister()
- ACL_LDAPDatabaseHandle()
- ACL_ListAclDelete()

- ACL_ListAppend()
- ACL_ListConcat()
- ACL_ListDestroy()
- ACL_ListFind()
- ACL_ListGetFirst()
- ACL_ListGetNameList()
- ACL_ListGetNext()
- ACL_ListNew()
- ACL_MethodFind()
- ACL_MethodGetDefault()
- ACL_MethodIsEqual()
- ACL_MethodNameIsEqual()
- ACL_MethodRegister()
- ACL_MethodSetDefault()
- ACL_ModuleRegister()
- ACL_NameListDestroy()
- ACL_ParseFile()
- ACL_ParseString()
- ACL_SetDefaultResult()
- ACL_SetupEval()
- ACL_WriteFile()
- ACL_WriteString()
- DbParseFn_t
- LASEvalFunc_t

- LASFlushFunc_t
- nserrDispose()
- nserrFAlloc()
- nserrFFree()
- nserrGenerate()
- PListAssignValue()
- PListCreate()
- PListDefProp()
- PListDeleteProp()
- PListDestroy()
- PListDuplicate()
- PListEnumerate()
- PListFindValue()
- PListFunc_t
- PListGetPool()
- PListGetValue()
- PListInitProp()
- PListNameProp()
- PListNew()
- PListSetType()
- PListSetValue()

ACLAttrGetterFn_t

`ACLAttrGetterFn_t` is the type definition for a function. This type describes a kind of callback function that obtains a value for an ACL attribute and enters the attribute and value into the subject property list.

You can register the function that you define by calling the `ACL_AttrGetterRegister()` function. Once this function and its attribute are registered, calling the `ACL_GetAttribute()` function will call your user-defined function.

Syntax

```
#include "aclapi.h"
NSAPI_PUBLIC
typedef int (*ACLAttrGetterFn_t)(NSErr_t *errp, PList_t subject,
   PList_t resource, PList_t auth_info, PList_t global_auth,
   void *arg);
```

Parameters Functions of this type have the following parameters:

`errp`	Pointer to the header for a list of error frames.
`subject`	Property list for the subject.
`resource`	Property list for the resource.
`auth_info`	Property list containing authentication information for the current attribute being evaluated.
`global_auth`	Property list containing authentication information for all attributes.
`arg`	Pointer to additional information.

Returns 0 if successful, or a negative value if unsuccessful.

Example See "LAS Code for the Email Attribute" and "LAS Code for the HTTP Referer Attribute".

See Also `ACL_AttrGetterRegister()`, `ACL_GetAttribute()`.

AclModuleInitFunc

`AclModuleInitFunc` is the type definition for a function. This type describes a kind of callback function that is initialized when the server starts up. (You can register this function by calling the `ACL_ModuleRegister()` function.)

Typically, you register the LAS in this function by calling the `ACL_LasRegister()`. You can also use this function to register "attribute getter" functions (by calling the `ACL_AttrGetterRegister()` function.)

Syntax

```
#include "aclapi.h"
NSAPI_PUBLIC
typedef int (*AclModuleInitFunc)(NSErr_t *errp);
```

Parameters Functions of this type have the following parameters:

`errp`	Pointer to the header for a list of error frames.

Returns 0 if successful, or a negative value if unsuccessful.

Example See "LAS Code for the Email Attribute" and "LAS Code for the HTTP Referer Attribute".

See Also `ACL_ModuleRegister()`, `ACL_LasRegister()`.

ACL_AclDestroy()

The `ACL_AclDestroy()` function frees an ACL from memory.

Syntax

```
#include "aclapi.h"
NSAPI_PUBLIC
void ACL_AclDestroy(NSErr_t *errp, ACLHandle_t *acl);
```

Parameters This function has the following parameters:

`errp`	Pointer to the header for a list of error frames.
`acl`	Pointer to the ACL that you want to free from memory.

Example The following example frees an ACL from memory.

```
#include "aclapi.h"
```

```
ACLHandle_t *acl;
NSErr_t err;

...

acl = ACL_AclNew(&err, "path=/usr/netscape/suitespot/docs/
protected.html");

... /* Work on ACL */...

ACL_AclDestroy(&err, acl);

...
```

See Also ACL_AclNew().

ACL_AclGetTag()

The `ACL_AclGetTag()` function gets the name (if any) for an ACL.

Syntax

```
#include "aclapi.h"
NSAPI_PUBLIC
const char *ACL_AclGetTag(ACLHandle_t *acl)
```

Parameters This function has the following parameters:

`acl`	Pointer to the ACL that you want to get the tag of.

Returns The name of the ACL or NULL if the ACL has no name.

Example The following example gets the name of an ACL.

```
#include "aclapi.h"

ACLHandle_t *acl;
const char *tag = 0;

...

/* Get the name of the ACL. */
tag = ACL_AclGetTag(acl);

...
```

See Also ACL_AclNew().

ACL_AclNew()

The `ACL_AclNew()` function creates a new ACL in memory.

When you are done using this ACL, you should free it from memory by calling the `ACL_AclDestroy()` function.

Syntax

```
#include "aclapi.h"
NSAPI_PUBLIC
ACLHandle_t *ACL_AclNew(NSErr_t *errp, char *tag);
```

Parameters This function has the following parameters:

`errp`	Pointer to the header for a list of error frames.
`tag`	Name identifying the new ACL (can be NULL)

Returns A new `ACLHandle_t`, which represents the new ACL, or NULL if an error occurs.

Example The following example creates a new ACL. The new ACL is identified by the name `path=/usr/netscape/suitespot/docs/protected.html`.

```
#include "aclapi.h"

ACLHandle_t *acl;
NSErr_t err;

...

acl = ACL_AclNew(&err, "path=/usr/netscape/suitespot/docs/
protected.html");

...
```

See Also `ACL_AclDestroy()`.

ACL_AttrGetterRegister()

The `ACL_AttrGetterRegister()` function registers a function used to get the value of an attribute. You call the `ACL_AttrGetterRegister()` function as part of the process of initializing your loadable authentication service (LAS).

When you call this function, you specify an "attribute getter" function that gets the value of an attribute by using a certain method and database. The attribute getter function is a function of type ACLAttrGetterFn_t.

When the ACL evaluator needs to retrieve the value of an attribute, it calls each matching attribute getter function for the attribute until one of them does not return LAS_EVAL_DECLINE.

An attribute getter function is matching if the current method and database type match the method and database type associated with the attribute getter function. (When you register an attribute getter function, you associate a method and database type with it.)

The method type ACL_METHOD_ANY matches any method. Similarly, the database type ACL_DBTYPE_ANY matches any database type. If you register an attribute getter function with ACL_METHOD_ANY and ACL_DBTYPE_ANY, the attribute getter function will always match.

Use the position argument to control the order in which the attribute getter functions are called. If you want an attribute getter function to be called first, use ACL_AT_FRONT as the position argument when registering the function.

Syntax

```
#include "aclapi.h"
NSAPI_PUBLIC
int ACL_AttrGetterRegister(NSErr_t *errp, const char *attr,
   ACLAttrGetterFn_t fn, ACLMethod_t m, ACLDbType_t d,
   int position, void *arg);
```

Parameters This function has the following parameters:

errp	Pointer to the header for a list of error frames.
attr	The attribute whose value can be retrieved by the function you plan to register.
fn	Pointer to the function that you want to register. This is a function that can get the value of the attribute specified by attr.
m	Method handle returned by the ACL_MethodFind() function or the ACL_MethodRegister() function. If you want the attribute getter function called for any method, specify ACL_METHOD_ANY as the method type.
d	Type of database used for authentication. If you want the attribute getter function called for any database type, specify ACL_DBTYPE_ANY as the database type.

`position`	Specifies the position of this function in the list of other functions that can get the attribute: • If `ACL_AT_FRONT`, the specified function becomes the first function in the list. • If `ACL_AT_END`, the specified function is appended to the end of the list.
`arg`	Additional pointer argument that you want passed to your function. When your function for retrieving attributes is called, this pointer is passed as the last argument of that function.

Returns 0 if successful, or a negative number if an error occurs.

Example The following example registers a function that gets the IP address for authentication. (The example uses `ACL_ATTR_IP` as the name of the IP attribute.) The example specifies the function's position as `ACL_AT_FRONT`.

For complete examples of registering "attribute getter" functions, see Chapter 7, "Examples of LAS Modules".

```
#include "aclapi.h"

MethodInfo_t methodType;
ACLDbType_t dbtype;
NSErr_t err;
int rv;

/* Register the My_LASIpGetter function as the first function */

if ((ACL_MethodFind(NULL, MY_METHOD, &methodType) == 0) &&
   (ACL_DbTypeFind(NULL, MY_DB_TYPE, &dbtype) == 0){

      rv = ACL_AttrGetterRegister(&err, ACL_ATTR_IP,
         My_LASIpGetter, methodType, dbtype, ACL_AT_FRONT, 0);
}

...
```

See Also ACLAttrGetterFn_t, ACL_GetAttribute().

ACL_AuthInfoGetDbname()

The `ACL_AuthInfoGetDbName()` function gets the name of the database from an `auth_info` property list.

Syntax

```
#include "aclapi.h"
```

```
NSAPI_PUBLIC
int ACL_AuthInfoGetDbname(PList_t auth_info, char **dbname);
```

Parameters This function has the following parameters:

`auth_info`	Property list of the authentication name/value pairs.
`dbname`	Pointer to the database name. If the `auth_info` property list doesn't have an entry for the name of the database, `dbname` is the default database name.

Returns 0 if successful.

Example The following example gets the name of the database from the `auth_info` property list.

```
#include "aclapi.h"

PList_t auth_info;
char *databaseName;
int rv;

...

rv = ACL_AuthInfoGetDbname(auth_info, &databaseName);

...
```

See Also ACL_AuthInfoGetDbType(), ACL_AuthInfoGetMethod(), ACL_DatabaseRegister(), ACL_DbTypeRegister(), ACL_ExprAddAuthInfo().

ACL_AuthInfoGetDbType()

The `ACL_AuthInfoGetDbType()` function gets the type of database used for authentication. The database type is specified in the `auth_info` property list.

Syntax

```
#include "aclapi.h"
NSAPI_PUBLIC
int ACL_AuthInfoGetDbType(NSErr_t *errp, PList_t auth_info,
   ACLDbType_t *t);
```

Parameters This function has the following parameters:

`errp`	Pointer to the header for a list of error frames.

`auth_info`	Property list of the authentication name/value pairs.
`t`	Pointer to the `ACLDbType_t` value specifying the type of database. If the `auth_info` property list doesn't have an entry for the name of the database, `t` is the default database type.

Returns 0 if successful.

Example The following example gets the database type from the `auth_info` property list.

```
#include "aclapi.h"

PList_t auth_info;
ACLDbType_t dbtype;
NSErr_t err;
int rv;

...

rv = ACL_AuthInfoGetDbType(&err, auth_info, &dbtype);

...
```

See Also ACL_AuthInfoGetDbname(), ACL_AuthInfoGetMethod(), ACL_DatabaseRegister(), ACL_DbTypeRegister().

ACL_AuthInfoGetMethod()

The `ACL_AuthInfoGetMethod()` function gets the method used for authentication. The method is specified in the `auth_info` property list.

Syntax

```
#include "aclapi.h"
NSAPI_PUBLIC
int ACL_AuthInfoGetMethod(NSErr_t *errp, PList_t auth_info,
   ACLMethod_t *t);
```

Parameters This function has the following parameters:

`errp`	Pointer to the header for a list of error frames.
`auth_info`	Property list of the authentication name/value pairs provided by the ACL_EvalTestRights() call to the LAS.
`t`	Pointer to the `ACLMethod_t` value representing the method used for authentication. If the `auth_info` property list doesn't have an entry for the method, t is the default method number.

Returns 0 if successful.

Example The following example gets the method used for authentication from the `auth_info` property list.

```
#include "aclapi.h"

PList_t auth_info;
ACLMethod_t auth_method;
NSErr_t err;
int rv;

...

rv = ACL_AuthInfoGetMethod(&err, auth_info, &auth_method);

...
```

See Also `ACL_AuthInfoGetDbname()`, `ACL_AuthInfoGetDbType()`, `ACL_DatabaseRegister()`, `ACL_DbTypeRegister()`.

ACL_CacheFlush()

The `ACL_CacheFlush()` function flushes the ACL cache.

Syntax

```
#include "aclapi.h"
NSAPI_PUBLIC
int ACL_CacheFlush(void);
```

Returns 0 if successful.

Example The following example flushes the ACL cache.

```
#include "aclapi.h"

...

ACL_CacheFlush();

...
```

ACL_CritEnter()

The `ACL_CritEnter()` function blocks other threads while the current thread is executing a critical section of code (a section of code that only one thread should be executing at a time).

The other threads remain blocked until the current thread calls the ACL_CritExit() function.

Syntax

```
#include "aclapi.h"
NSAPI_PUBLIC
void ACL_CritEnter(void);
```

Example The following simplified example blocks all other threads from executing until the ACL_CritExit() function is called. This prevents other threads from setting the value of the myvar variable.

```
#include "aclapi.h"

int myvar;

...

/* Block all other threads until you are done with this section. */

ACL_CritEnter();

/* You can call functions here to modify data. Other threads are blocked
until the current thread executes ACL_CritExit(). */

myvar = 1;

/* When done with this critical section of code, call ACL_CritExt() to
unblock the other threads.*/

ACL_CritExit();

...
```

See Also ACL_CritExit().

ACL_CritExit()

Call the ACL_CritExit() function to unblock other threads when the current thread is done executing a critical section of code (a section of code that only one thread should be executing at a time).

At the start of the critical section of code, the thread should call the ACL_CritEnter() function to block the other threads.

Syntax

```
#include "aclapi.h"
NSAPI_PUBLIC
void ACL_CritExit(void);
```

Example See "ACL_CritEnter()".

See Also `ACL_CritEnter()`.

ACL_DatabaseFind()

The `ACL_DatabaseFind()` function finds an authentication database with a specified name and gets the database type and a pointer to the database. The database must be registered (by calling the `ACL_DatabaseRegister()` function) before you can find the database. (In other words, `ACL_DatabaseFind()` does not find unregistered databases.)

Syntax

```
#include "aclapi.h"
NSAPI_PUBLIC
int ACL_DatabaseFind(NSErr_t *errp, const char *dbname,
   ACLDbType_t *dbtype, void **db);
```

Parameters This function has the following parameters:

`errp`	Pointer to the header for a list of error frames.
`dbname`	Name of the registered database that you want to find.
`dbtype`	Pointer to the type of the database.
`db`	Pointer to the data retrieved by the parsing function. The parsing function retrieves this data from the database URL when the database is registered during an `ACL_DatabaseRegister()` function call.

Returns `LAS_EVAL_TRUE` if successful, or `LAS_EVAL_FAILURE` if an error occurred.

Example The following example uses the `ACL_DatabaseFind()` function to determine if a database has already been registered.

```
#include "aclapi.h"

NSErr_t err;
const char *url;
ACLDbType_t dbtype;
void *db;

...

/* If the database is already registered, don't do anything.

if (ACL_DatabaseFind(&err, url, &dbtype, &db) == LAS_EVAL_TRUE) {
   return 0;
}

...
```

See Also `ACL_DatabaseRegister()`.

ACL_DatabaseGetDefault()

The `ACL_DatabaseGetDefault()` function returns you the name of the default database, which is normally the database named "default." You can change the default database in the obj.conf file, see “Setting the Default Database”. You can also use the ACL_DatabaseSetDefault() function to change the default database.

Syntax const char *ACL_DatabaseGetDefault(NSErr_t *errp);

Parameters This function has the following parameters:

`errp`	Pointer to the header for a list of error frames.

Returns Returns the name of the default database

See Also `ACL_DatabaseSetDefault()`

ACL_DatabaseRegister()

The `ACL_DatabaseRegister()` function registers an authentication database by name. The act of registering a database stores information about the database, including its name, type, and a pointer to the database. You can call the `ACL_DatabaseFind()` function to get this information, based on the name of the database.

Syntax

```
#include "aclapi.h"
NSAPI_PUBLIC
int ACL_DatabaseRegister (NSErr_t *errp, ACLDbType_t dbtype,
   const char *dbname, const char *url, PList_t plist);
```

Parameters This function has the following parameters:

`errp`	Pointer to the header for a list of error frames.
`dbtype`	Database type, which must be registered. This function uses the database type to find the corresponding parsing function, which is called to retrieve data by parsing the database URL.
`dbname`	Name of the database that you want to register.

`url`	Database URL to be parsed by the parsing function for this database type.
`plist`	Property list passed to the parsing function for this database type.

Returns 0 if successful, or a negative number if an error occurred.

Example The following example registers the database named `mydb`.

When the `ACL_DatabaseRegister()` function is called, the `url` argument (the reference to the database) is automatically parsed by the parsing function associated with the database type `dbtype`. (See `ACL_DbTypeRegister()` and `ACL_DbTypeParseFn()` for more information on the parsing function.)

The result of the parsing function (the parsed reference) is stored in a hash table and is associated with the database name. You can get this result by calling the `ACL_DatabaseFind()` function.

```
#include "aclapi.h"

NSErr_t err;
ACLDbType_t dbtype;
const char *dbname = "mydb";
const char *url;
PList_t plist;
int rv;

...

/* Register the database. */

rv = ACL_DatabaseRegister(&err, dbtype, dbname, url, plist);

...
```

See Also .`ACL_DatabaseFind()`, `ACL_DbTypeRegister()`.

ACL_DatabaseSetDefault()

The `ACL_DatabaseSetDefault()` function specifies the database used as the default authentication database.

Syntax

```
#include "aclapi.h"
NSAPI_PUBLIC
int ACL_DatabaseSetDefault(NSErr_t *errp, const char *dbname);
```

Parameters This function has the following parameters:

`errp`	Pointer to the header for a list of error frames.
`dbname`	Name of the database you want to set as the default database.

Returns 0 if successful.

Example The following example sets the default database to the database named `mydb`.

```
#include "aclapi.h"

NSErr_t err;
int rv;

...

rv = ACL_DatabaseSetDefault(&err, "mydb");

...
```

See Also `ACL_DatabaseFind()`, `ACL_DatabaseRegister()`, `ACL_DatabaseGetDefault()`.

ACL_DbTypeFind()

The `ACL_DbTypeFind()` function finds a database type with a specified name. The database type must be registered (by calling the `ACL_DbTypeRegister()` function) before you can find the database type. (In other words, `ACL_DbTypeFind()` does not find unregistered database types.)

Syntax

```
#include "aclapi.h"
NSAPI_PUBLIC
int ACL_DbTypeFind(NSErr_t *errp, const char *name,
   ACLDbType_t *t);
```

Parameters This function has the following parameters:

`errp`	Pointer to the header for a list of error frames.
`name`	Name of the database type that you want to find.
`t`	Pointer to a buffer to hold the database type returned by this function.

Returns 0 if successful, or a negative number if an error occurred.

Example The following example finds the database type for the type named `ldap`.

```
#include "aclapi.h"

NSErr_t err;
const char *name = "ldap";
ACLDbType_t dbtype;
int rv;

...

/* Find the database type for the type named "ldap". */
rv = ACL_DbTypeFind(&err, name, &dbtype);

...
```

See Also .ACL_DbTypeRegister().

ACL_DbTypeGetDefault()

The `ACL_DbTypeGetDefault()` function gets the database type that is used as the default database type.

Syntax

```
#include "aclapi.h"
NSAPI_PUBLIC
ACLDbType_t ACL_DbTypeGetDefault(NSErr_t *errp);
```

Parameters This function has the following parameters:

`errp`	Pointer to the header for a list of error frames.

Returns The default database type.

Example The following example gets the default database type.

```
#include "aclapi.h"

NSErr_t err;
ACLDbType_t dbtype;

...

/* Get the default database type. */
dbtype = ACL_DbTypeGetDefault(&err);

...
```

See Also

ACL_DbTypeIsEqual()

The `ACL_DbTypeIsEqual()` function compares two database types and determines if the types are equal.

Syntax

```
#include "aclapi.h"
NSAPI_PUBLIC
int ACL_DbTypeIsEqual(NSErr_t *errp, const ACLDbType_t t1,
   const ACLDbType_t t2);
```

Parameters This function has the following parameters:

`errp`	Pointer to the header for a list of error frames.
`t1`	One of the two database types that you want to compare.
`t2`	The other database type that you want to compare.

Returns 1 if the database types are equal, 0 if not.

Example The following example logs a message if two database types are not equal.

```
#include "aclapi.h"

NSErr_t err;
ACLDbType_t dbtype1, dbtype2;

...

/* Check to see if the dbtypes are equal. */

if (!ACL_DbTypeIsEqual(&err, dbtype1, dbtype2)) {
   log_error(LOG_INFORM, "my_function", NULL, NULL,
      "Database types not equal");
}

...
```

See Also `ACL_DbTypeNameIsEqual()`.

ACL_DbTypeIsRegistered()

The `ACL_DbTypeIsRegistered()` function determines whether the specified database type is registered.

Syntax

```
#include "aclapi.h"
NSAPI_PUBLIC
```

```
int ACL_DbTypeIsRegistered (NSErr_t *errp, const ACLDbType_t t);
```

Parameters This function has the following parameters:

`errp`	Pointer to the header for a list of error frames.
`t`	Database type that you want to check.

Returns 1 if the database type is registered, 0 if not.

Example The following example checks to see if a database type has been registered.

```
#include "aclapi.h"

NSErr_t err;
ACLDbType_t dbtype;

...

/* If database type is already registered, log a message. */

if (ACL_DbTypeIsRegistered(&err, dbtype)) {
   log_error(LOG_INFORM, "my_function", NULL, NULL,
      "Database type %d is already registered", dbtype);


...
```

See Also `ACL_DbTypeRegister()`.

ACL_DbTypeNameIsEqual()

The `ACL_DbTypeNameIsEqual()` function determines whether a database type has a given name. The database type name is assigned to a database type when that type is registered. (See the `ACL_DbTypeRegister()` function for details.)

Syntax

```
#include "aclapi.h"
NSAPI_PUBLIC
int ACL_DbTypeNameIsEqual(NSErr_t *errp, const ACLDbType_t t1,
   const char *name);
```

Parameters This function has the following parameters:

`errp`	Pointer to the header for a list of error frames.

`t1`	The database type that you want to compare against the specified type name.
`name`	Type name to compare against the type name of the specified database type `t1`.

Returns 1 if the database types have matching names, 0 if not.

Example The following example checks to see if the database type `dbtype` has the name `ldap`.

```
#include "aclapi.h"

ACLDbType_t dbtype;
NSErr_t err;

...

/* If the database type is not ldap, log a message. */

if (!ACL_DbTypeNameIsEqual(&err, dbtype, "ldap")) {
   log_error(LOG_INFORM, "my_function", NULL, NULL,
      "Database type is not ldap.\n");
}

...
```

See Also `ACL_DbTypeRegister()`, `ACL_DbTypeIsEqual()`.

ACL_DbTypeParseFn()

The `ACL_DbTypeParseFn()` function returns the parsing function associated with the specified database type.

A database type and its corresponding parsing function are associated with each other when the database type is registered (by calling the `ACL_DbTypeRegister()` function).

The parsing function for a database type is a function that parses the database URL and retrieves relevant data from the URL. For example, for the `ldap` database type, the corresponding parsing function parses LDAP URLs for information such as the host name, port number, and base DN.

For more information on parsing functions and database types, see "ACL_DbTypeRegister()".

Syntax

```
#include "aclapi.h"
NSAPI_PUBLIC DbParseFn_t
ACL_DbTypeParseFn(NSErr_t *errp, const ACLDbType_t dbtype);
```

Parameters This function has the following parameters:

`errp`	Pointer to the header for a list of error frames.
`dbtype`	Database type for which you want to retrieve the parsing function.

Returns The parsing function for the specified database type, or 0 if the database type is not registered.

Example The following example retrieves and calls the parsing function for a database type.

```
#include "aclapi.h"

DbParseFn_t parseFunc;
ACLDbType_t dbtype;
NSErr_t err;
int rv;
const char *dbname;
const char *url;
PList_t plist;
void *db;

...

/* Get the parsing function for the specified type of database. */
parseFunc = ACL_DbTypeParseFn(&err, dbtype);

/* Call the parsing function. */
rv = (*parseFunc)(&err, dbtype, dbname, url, plist, (void **)&db);

...
```

See Also `ACL_DbTypeRegister()`.

ACL_DbTypeRegister()

The `ACL_DbTypeRegister()` function registers a database type by name and associates a parsing function with the database type. The parsing function must be a function of the type `DbParseFn_t`.

The purpose of the parsing function is to retrieve relevant data from the database URL.

For example, one of the standard database types is `ldap`, which represents an LDAP directory. The parsing function for this database type parses this type of URL:

```
ldap://directory.aceindustry.com:390/o=Ace Industry,c=US
```

and retrieves the relevant information such as the host name, port, and base DN. The parsed data is made available through the `db` argument of the parsing function (`db` is a pointer to the handle of this structure).

The parsing function is called whenever you call `ACL_DatabaseRegister()` to register a database of this type. `ACL_DatabaseRegister()` associates the parsed data with the database name. The parsed data can be retrieved later by calling the `ACL_DatabaseFind()` function.

Syntax

```
#include "aclapi.h"
NSAPI_PUBLIC
int ACL_DbTypeRegister(NSErr_t *errp, const char *name,
   DbParseFn_t func, ACLDbType_t *t);
```

Parameters This function has the following parameters:

`errp`	Pointer to the header for a list of error frames.
`name`	Database type as a string name (for example, "ldap").
`func`	Function used to parse the database URL and retrieve a pointer to the parsed data.
`t`	Pointer to the number assigned to the database type when it is registered. Each database type is assigned a unique number.

Returns 0 if successful, or a negative number if an error occurred.

Example The following example illustrates how the `ldap` database type is registered. The function for parsing LDAP URLs is named `parse_ldap_url`.

```
#include "aclapi.h"

ACLDbType_t ACL_DbTypeLdap;
NSErr_t err;
int rv;

...

/* Register the ldap database type */

rv = ACL_DbTypeRegister(&err, ACL_DBTYPE_LDAP, parse_ldap_url,
&ACL_DbTypeLdap);
```

```
...
```

See Also DbParseFn_t, ACL_DatabaseRegister(), ACL_DatabaseFind().

ACL_EvalDestroy()

The ACL_EvalDestroy() function frees an ACL evaluation context from memory. This function also frees any subject property lists, resource property lists, and lists of ACLs associated with the context (provided that there are no other references to these lists).

Call this function after you are done working with an ACL evaluation context that you have allocated with ACL_EvalNew().

Syntax

```
#include "aclapi.h"
NSAPI_PUBLIC
void ACL_EvalDestroy(NSErr_t *errp, pool_handle_t *pool,
   ACLEvalHandle_t *acleval);
```

Parameters This function has the following parameters:

errp	Pointer to the header for a list of error frames.
pool	Pointer to the memory pool associated with the evaluation context. This is the memory pool that was passed into the ACL_EvalNew() function.
acleval	Pointer to the ACL evaluation context that you want to free from memory.

Example The following example frees an evaluation context from memory after it is no longer needed.

```
#include "aclapi.h"

ACLEvalHandle_t *acleval;
ACLListHandle_t *acllist;
static char *rights[] = {"http_get", NULL};
char *bong;
char *bong_type;
char *acl_tag;
int  expr_num;
int  rv;

...
```

```
/* Allocate memory for a new evaluation context. */
acleval = ACL_EvalNew(NULL, NULL);

/* Associate an ACL list with the context. */
ACL_EvalSetACL(NULL, acleval, acllist);

/* Determine the rights to a resource. */
rv = ACL_EvalTestRights(NULL, acleval, rights, http_generic,
   &bong_type, &bong, &acl_tag, &expr_num);

/* Free the evaluation context when done. */
ACL_EvalDestroy(NULL, acleval);

...
```

See Also ACL_EvalNew().

ACL_EvalGetResource()

The `ACL_EvalGetResource()` function gets the resource property list associated with an evaluation context.

If you want to set the resource property list of an evaluation context, call the `ACL_EvalSetResource()` function.

Syntax

```
#include "aclapi.h"
NSAPI_PUBLIC
PList_t ACL_EvalGetResource(NSErr_t *errp,
   ACLEvalHandle_t *acleval);
```

Parameters This function has the following parameters:

`errp`	Pointer to the header for a list of error frames.
`acleval`	Pointer to the ACL evaluation context for which you want to get the resource property list.

Returns The property list for the resource in this evaluation context.

Example The following example gets the value of the realm (the string presented to the user during authentication) from the resource property list in an evaluation context.

```
#include "aclapi.h"

NSErr_t err;
ACLEvalHandle_t *acleval;
```

```
...

/* Get the resource property list associated with this context. */
PList_t resource = ACL_EvalGetResource(&err, acleval);

/* Get the realm/prompt from the resource list. */
int auth_req = PListGetValue(resource, ACL_ATTR_WWW_AUTH_PROMPT_INDEX,
   (void **)&prompt, NULL);

...
```

See Also ACL_EvalSetResource().

ACL_EvalGetSubject()

The `ACL_EvalGetSubject()` function gets the subject property list associated with an evaluation context.

If you want to set the subject property list of an evaluation context, call the ACL_EvalSetSubject() function.

Syntax

```
#include "aclapi.h"
NSAPI_PUBLIC
PList_t ACL_EvalGetSubject(NSErr_t *errp,
   ACLEvalHandle_t *acleval);
```

Parameters This function has the following parameters:

`errp`	Pointer to the header for a list of error frames.
`acleval`	Pointer to the ACL evaluation context for which you want to get the subject property list.

Returns The property list for the subject in this evaluation context.

Example The following example gets the subject property list from an evaluation context.

```
#include "aclapi.h"

NSErr_t err;
ACLEvalHandle_t *acleval;

...

/* Get the subject property list associated with this context. */
PList_t subject = ACL_EvalGetSubject(&err, acleval);

...
```

See Also ACL_EvalSetSubject().

ACL_EvalNew()

The ACL_EvalNew() function creates a new context for evaluating an ACL. An ACL evaluation context is used to save the ACL evaluation state for a particular subject's access to a resource.

An ACL evaluation context associates an ACL with a subject property list and a resource property list. You use this context to determine whether a subject has the appropriate access privileges to a resource.

When you are done using an ACL evaluation context, you should free it from memory by calling the ACL_EvalDestroy() function.

Syntax

```
#include "aclapi.h"
NSAPI_PUBLIC
ACLEvalHandle_t *ACL_EvalNew(NSErr_t *errp,
   pool_handle_t *pool);
```

Parameters This function has the following parameters:

errp	Pointer to the header for a list of error frames.
pool	Pointer to a memory pool. If you pass a nonnull pointer, the server uses this pool to allocate memory for the evaluation context.

Returns A new ACLEvalHandle_t, which represents the new context for evaluating an ACL.

Example The following example allocates memory for a new evaluation context. This context is used to determine if the specified user has access rights to the specified resource.

```
#include "aclapi.h"

ACLEvalHandle_t *acleval;
ACLListHandle_t *acllist;
PList_t subject;
NSErr_t err;
int rights_rv;
int rv;
char *testRight;
char *deny = NULL;
char *deny_generic = NULL;
```

```
char *acl_tag;
int expr_num;

...

/* Create an ACL evaluation context. */
acleval = ACL_EvalNew(&err, NULL);

/* Specify the subject property list for this context. */
rv = ACL_EvalSetSubject(&err, acleval, subject);

/* Specify the ACL list for this context. */
rv = ACL_EvalSetACL(&err, acleval, acllist);

/* Test the rights of the user. */
rights_rv = ACL_EvalTestRights (&err, acleval, &testRight,
   map_generic, &deny, &deny_generic, &acl_tag, &expr_num);

...
```

See Also ACL_EvalDestroy().

ACL_EvalSetACL()

The `ACL_EvalSetACL()` function sets the ACL list associated with an evaluation context.

Syntax

```
#include "aclapi.h"
NSAPI_PUBLIC
int ACL_EvalSetACL(NSErr_t *errp, ACLEvalHandle_t *acleval,
   ACLListHandle_t *acllist);
```

Parameters This function has the following parameters:

`errp`	Pointer to the header for a list of error frames.
`acleval`	Pointer to the ACL evaluation context for which you want to set the ACL list.
`acllist`	Pointer to the ACL list that you want to associate with the ACL evaluation context.

Returns 0 if successful.

Example The following example associates an ACL list with an evaluation context.

```
#include "aclapi.h"
```

```
ACLEvalHandle_t *acleval;
ACLListHandle_t *acllist;
NSErr_t err;
int rv;

...

/* Create an ACL evaluation context. *
acleval = ACL_EvalNew(&err, NULL);

/* Associate an ACL list with the context. */
rv = ACL_EvalSetACL(&err, acleval, acllist);

...
```

See Also ACL_EvalNew(), ACL_ListNew().

ACL_EvalSetResource()

The `ACL_EvalSetResource()` function sets the resource property list associated with an evaluation context. This list contains information about the resource that can be used to help determine whether the subject is allowed access to the resource.

If you want to get the resource property list of an evaluation context, call the ACL_EvalGetResource() function.

Syntax

```
#include "aclapi.h"
NSAPI_PUBLIC
int ACL_EvalSetResource(NSErr_t *errp, ACLEvalHandle_t *acleval,
   PList_t resource);
```

Parameters This function has the following parameters:

`errp`	Pointer to the header for a list of error frames.
`acleval`	Pointer to the ACL evaluation context for which you want to set the resource property list.
`resource`	The resource property list that you want to associate with the ACL evaluation context.

Returns 0 if successful.

Example The following example sets the resource property list for an evaluation context.

```
#include "aclapi.h"
```

```
PList_t resource;
ACLEvalHandle_t *acleval;
char *request;
NSErr_t err;
int rv;

...

/* Allocate memory for a new property list for the resource.
resource = PListCreate(NULL, ACL_ATTR_INDEX_MAX, 0, 0);

/* Specify the request itself as one of the properties. */
rv = PListInitProp(resource, ACL_ATTR_REQUEST_INDEX,
   ACL_ATTR_REQUEST, request, NULL);

/* Add this property list to the evaluation context. */
rv = ACL_EvalSetResource(&err, acleval, resource);

...
```

See Also ACL_EvalGetResource().

ACL_EvalSetSubject()

The `ACL_EvalSetSubject()` function sets the subject property list associated with an evaluation context. This list contains information about the subject that can be used to help determine whether the subject is allowed access to the resource.

If you want to get the subject property list of an evaluation context, call the ACL_EvalGetSubject() function.

Syntax

```
#include "aclapi.h"
NSAPI_PUBLIC
int ACL_EvalSetSubject(NSErr_t *errp, ACLEvalHandle_t *acleval,
   PList_t subject);
```

Parameters This function has the following parameters:

`errp`	Pointer to the header for a list of error frames.
`acleval`	Pointer to the ACL evaluation context for which you want to set the subject property list.
`subject`	The subject property list that you want to associate with the ACL evaluation context.

Returns 0 if successful.

Example The following example creates a new subject property list, adds the subject's IP address to the list, and associates the list with an evaluation context.

```
#include "aclapi.h"

ACLEvalHandle_t *acleval;
PList_t subject;
int rv;
NSErr_t err;
char *ipaddr;

/* Create a new evaluation context. */
acleval = ACL_EvalNew(&err, NULL);

/* Create a new subject list. */
subject = PListCreate(NULL, ACL_ATTR_INDEX_MAX, 0, 0);

/* Add the subject's IP address to the property list. */
rv = PListInitProp(subject, ACL_ATTR_IP_INDEX, ACL_ATTR_IP,
   ipaddr, NULL);

/* Add this subject property list to the evaluation context. */
rv = ACL_EvalSetSubject(&err, acleval, subject);

...
```

See Also ACL_EvalGetSubject().

ACL_EvalTestRights()

The `ACL_EvalTestRights()` function determines whether a given subject is allowed a set of access rights to a given resource.

To test the access rights, you need to specify the ACL evaluation context, the rights requested, and a mapping of specific-to-generic rights. For access to Netscape web servers, you can use `http_generic` as the mapping.

`http_generic` maps rights specific to Netscape web servers to generic rights. The following table lists this mapping:

Table 9.13 How http_generic maps specific rights to generic rights

Generic Right	Web Server Rights

`read`	`http_get`
	`http_head`
	`http_trace`
	`http_revlog`
	`http_options`
	`http_getattribute`
	`http_index`
	`http_getproperties`
	`http_getattributenames`

`write`	`http_put`
	`http_mkdir`
	`http_startrev`
	`http_stoprev`
	`http_edit`
	`http_copy`
	`http_move`
	`http_post`
	`http_save`
	`http_setattribute`
	`http_revadd`
	`http_revlabel`
	`http_lock`
	`http_unlock`
	`http_unedit`
	`http_stoprev`
	`http_startrev`

`execute`	`http_post`
`delete`	`http_delete` `http_destroy`

`info`	`http_head`
	`http_trace`
	`http_options`

`list`	`http_index`

For the `rights` argument, you can pass an array of rights for Netscape web servers (for example, `{"http_get", "http_post", NULL}`).

Syntax `#include "aclapi.h"`

```
NSAPI_PUBLIC
int ACL_EvalTestRights(NSErr_t *errp, ACLEvalHandle_t *acleval,
   char **rights, char **map_generic, char **deny_type,
   char **deny_response, char **acl_tag, int *expr_num);
```

Parameters This function has the following parameters:

`errp`	Pointer to the header for a list of error frames.
`acleval`	Pointer to the ACL evaluation context used to evaluate the subject's rights to the resource.
`rights`	Array of strings specifying the requested rights (for example, `{"http_get", ... }`).
`map_generic`	Array of strings listing the specific rights that map to the generic rights (for example, `http_generic`).
`deny_type`	Specifies the type of URL returned to the user who failed to access the resource. This type can be one of the following: • `url` Choose this type if `deny_response` specifies the absolute URL that the user is redirected to. • `uri` Choose this type if `deny_response` specifies a partial URL that the user should be redirected to.
`deny_response`	Pointer to the partial or absolute URL to which the user is redirected.
`acl_tag`	If the subject is denied access to the resource, pointer to the name of the ACL that denied access.
`expr_num`	If the subject is denied access to the resource, pointer to the number of the ACE within the ACL that denied access.

Returns Returns one of the following values:

- `ACL_RES_ALLOW` if the subject has the specified rights to the resource.
- `ACL_RES_DENY` if the subject does not have the specified rights to the resource.
- `ACL_RES_FAIL` if an error occurs.

- `ACL_RES_INVALID` if a configuration error occurs (for example, if an invalid argument was passed to the evaluation function).

Note As each ACE is processed, when a new authentication statement is encountered, the function deletes any entries in the subject property list and with names that match the attributes in the authentication statement. This prevents the improper caching of attribute values across different authentication databases or methods.

For example, if the authentication statement begins with "authenticate (user,group)...", the properties "user" and "group" are deleted from the subject property list.

Example The following example determines if a subject has the rights to access a resource.

```
#include "aclapi.h"

ACLEvalHandle_t *acleval;
ACLListHandle_t *acllist;
PList_t subject, resource;
NSErr_t err;
int rv;
char **rights = {"http_get", "http_put", NULL};
char *deny_type;
char *deny_response;
char *acl_tag;
int expr_num;

...

/* Create a new evaluation context. */
acleval = ACL_EvalNew(&err, NULL);

/* Add the subject property list to the evaluation context. */
rv = ACL_EvalSetSubject(&err, acleval, subject);

/* Add the resource property list to the evaluation context. */
rv = ACL_EvalSetResource(&err, acleval, resource);

/* Associate an ACL list with the context. */
rv = ACL_EvalSetACL(&err, acleval, acllist);

/* Determine if the subject can access the resource. */
rv = ACL_EvalTestRights(&err, acleval, rights, http_generic,
   &deny_type, &deny_response, &acl_tag, &expr_num);

...
```

See Also ACL_EvalNew(), ACL_EvalSetACL(), ACL_EvalSetResource(), ACL_EvalSetSubject().

ACL_ExprAddArg()

The `ACL_ExprAddArg()` function adds an access right (for example, `read`, `write`, `execute`, `list`, or `info`) to an expression. In terms of ACL file syntax, calling this function is equivalent to adding rights to an `allow` or `deny` statement.

This function applies only to expressions of type `ACL_EXPR_TYPE_ALLOW` or `ACL_EXPR_TYPE_DENY`.

Syntax

```
#include "aclapi.h"
NSAPI_PUBLIC
int ACL_ExprAddArg(NSErr_t *errp, ACLExprHandle_t *expr,
   char *arg);
```

Parameters This function has the following parameters:

`errp`	Pointer to the header for a list of error frames.
`expr`	Pointer to the attribute expression (of type `ACL_EXPR_TYPE_ALLOW` or `ACL_EXPR_TYPE_DENY`) that you want to modify.
`arg`	Access right that you want to add to the expression (for example, `read`, `write`, `execute`, `list`, or `info`).

Returns 0 if successful, or a negative number if an error occurred.

Note This function is part of a low-level interface that may be useful for defining ACLs without using the ACL file syntax. Because this is a low-level interface, future support of this interface is not guaranteed. Wherever possible, you should use the `ACL_ParseFile()` and `ACL_ParseString()` functions instead.

Example The following example defines an attribute expression in which the user `guest` is denied read and write access to the resource.

```
#include "aclapi.h"

ACLExprHandle_t *expr;
NSErr_t err;
int rv;

...

/* Create a new expression that "denies" access. */
expr = ACL_ExprNew(ACL_EXPR_TYPE_DENY);
```

```
/* Deny "read" and "write" privileges. */
rv = ACL_ExprAddArg(&err, expr, "read");
rv = ACL_ExprAddArg(&err, expr, "write");

/* Specify the terms of the restriction. */
rv = ACL_ExprTerm(&err, expr, "user", CMP_OP_EQ, "guest");

...
```

The example above is equivalent to adding the following statements to an ACL file:

```
deny (read, write)
   user = "guest";
```

See Also ACL_ExprNew().

ACL_ExprAddAuthInfo()

The `ACL_ExprAddAuthInfo()` function adds authentication information to an expression. In terms of ACL file syntax, calling this function is equivalent to specifying the method, database, and other information in an `authenticate` statement.

This function applies only to expressions of type `ACL_EXPR_TYPE_AUTH`.

One of the input parameters to this function is a property list containing the authentication information. You need to create this property list yourself. For example, suppose you wanted to specify the equivalent of the following authentication statement in an ACL file:

```
authenticate (user,group) {
   method="basic";
   database="ldapdb";
};
```

To define this statement programmatically, you would create a new property list with name-value pairs for `method` and `database`:

```
PList_t auth_info = PListNew(NULL);
ACLMethod_t methodType;
ACLDbType_t dbType;

...

if (auth_info) {
   PListInitProp(auth_info, 0, ACL_ATTR_METHOD, methodType, NULL);
   PListInitProp(auth_info, 0, ACL_ATTR_DATABASE, "ldapdb", NULL);
```

```
   PListInitProp(auth_info, 0, ACL_ATTR_DBTYPE, dbType, NULL);
}
```

After creating this property list, you can pass it as an argument to the `ACL_ExprAddAuthInfo()` function.

Syntax

```
#include "aclapi.h"
NSAPI_PUBLIC
int ACL_ExprAddAuthInfo(ACLExprHandle_t *expr,
   PList_t auth_info);
```

Parameters This function has the following parameters:

`expr`	Pointer to the attribute expression (of type `ACL_EXPR_TYPE_AUTH`) for which you want to specify authentication information.
`auth_info`	The authentication information to add to the attribute expression. This is a property list (`PList_t`) that you need to create yourself.

Returns 0 if successful, or a negative number if an error occurred.

Note This function is part of a low-level interface that may be useful for defining ACLs without using the ACL file syntax. Because this is a low-level interface, future support of this interface is not guaranteed. Wherever possible, you should use the `ACL_ParseFile()` and `ACL_ParseString()` functions instead.

Example The following example specifies that the basic method and the LDAP user database are used for authentication.

```
#include "aclapi.h"

ACLExprHandle_t *expr;
PList_t auth_info;
NSErr_t err;
int rv;
ACLMethod_t methodType;
ACLDbType_t dbType;

...

/* Create a new property list specifying authentication information. */
auth_info = PListNew(NULL);

/* Add the authentication specifics to the property list. */
if (auth_info) {
   PListInitProp(auth_info, 0, ACL_ATTR_METHOD, methodType, NULL);
   PListInitProp(auth_info, 0, ACL_ATTR_DATABASE, "ldapdb", NULL);
   PListInitProp(auth_info, 0, ACL_ATTR_DBTYPE, dbType, NULL);
}
```

```
/* Create a new authentication expression. */
expr = ACL_ExprNew(ACL_EXPR_TYPE_AUTH);

/* Specify the attributes that require authentication. */
rv = ACL_ExprAddArg(&err, expr, "user");
rv = ACL_ExprAddArg(&err, expr, "group");

/* Add the authentication information to the expression. */
rv = ACL_ExprAddAuthInfo(expr, auth_info);

...
```

The example above is equivalent to adding the following statements to an ACL file:

```
authenticate (user,group) {
   method="basic";
   database="ldapdb";
};
```

See Also ACL_ExprNew(), ACL_ExprAddArg().

ACL_ExprAnd()

The `ACL_ExprAnd()` function uses a logical AND operator to combine the preceding two terms or subexpressions in an attribute expression. In terms of ACL file syntax, calling this function is equivalent to using `and` to combine two terms in an `allow` or `deny` statement.

This function applies only to expressions of type `ACL_EXPR_TYPE_ALLOW` or `ACL_EXPR_TYPE_DENY`.

Syntax

```
#include "aclapi.h"
NSAPI_PUBLIC
int ACL_ExprAnd(NSErr_t *errp, ACLExprHandle_t *acl_expr);
```

Parameters This function has the following parameters:

`errp`	Pointer to the header for a list of error frames.
`acl_expr`	Pointer to the attribute expression (of type `ACL_EXPR_TYPE_ALLOW` or `ACL_EXPR_TYPE_DENY`) that you want to modify.

Returns 0 if successful, or a negative number if an error occurred.

Note This function is part of a low-level interface that may be useful for defining ACLs without using the ACL file syntax. Because this is a low-level interface, future support of this interface is not guaranteed. Wherever possible, you should use the `ACL_ParseFile()` and `ACL_ParseString()` functions instead.

Example The following example defines an attribute expression in which read access to a resource is allowed if the subject is the user `guest` and if the subject's client machine is in the domain `netscape.com`.

```
#include "aclapi.h"

ACLExprHandle_t *expr;
NSErr_t err;
int rv;

...

/* Create a new expression that "allows" access. */
expr = ACL_ExprNew(ACL_EXPR_TYPE_ALLOW);

/* Specify that you want to allow "read" access. */
rv = ACL_ExprAddArg(&err, expr, "read");

/* Specify that this applies to users named "guest" who access the
resource from clients within the netscape.com domain. */

rv = ACL_ExprTerm(&err, expr, "user", CMP_OP_EQ, "guest");
rv = ACL_ExprTerm(&err, expr, "dns", CMP_OP_EQ, "*.netscape.com");

/* Combine the last two terms so both must be true to allow access. */
rv = ACL_ExprAnd(&err, expr);

...
```

The example above is equivalent to adding the following statements to an ACL file:

```
allow (read)
   user = "guest" and dns = "*.netscape.com";
```

See Also `ACL_ExprNot()`, `ACL_ExprOr()`, `ACL_ExprTerm()`.

ACL_ExprAppend()

The `ACL_ExprAppend()` function appends the specified attribute expression to an ACL. You can call this function to build an ACL or to add to an ACL.

Syntax

```
#include "aclapi.h"
NSAPI_PUBLIC
```

```
int ACL_ExprAppend(NSErr_t *errp, ACLHandle_t *acl,
   ACLExprHandle_t *expr);
```

Parameters This function has the following parameters:

`errp`	Pointer to the header for a list of error frames.
`acl`	Pointer to the ACL that you want to add the expression to.
`expr`	Pointer to the attribute expression that you want to add to the ACL.

Returns If successful, returns the number of expressions in the ACL after the expression has been appended. If unsuccessful, returns a negative number.

Note This function is part of a low-level interface that may be useful for defining ACLs without using the ACL file syntax. Because this is a low-level interface, future support of this interface is not guaranteed. Wherever possible, you should use the `ACL_ParseFile()` and `ACL_ParseString()` functions instead.

Example The following example creates an ACL that contains two attribute expressions. The example calls the `ACL_ExprAppend()` function to add the expressions to the newly created ACL.

```
#include "aclapi.h"

ACLHandle_t *acl;
ACLExprHandle_t *expr, *expr2;
NSErr_t err;
int rv;

/* Create a new ACL. */
acl = ACL_AclNew(&err, "My New ACL");

/* Create an expression to deny write access to users named guest. */
expr = ACL_ExprNew(ACL_EXPR_TYPE_DENY);
rv = ACL_ExprAddArg(&err, expr, "write");
rv = ACL_ExprTerm(&err, expr, "user", CMP_OP_EQ, "guest");

/* Append the expression to the ACL. */
rv = ACL_ExprAppend(&err, acl, expr);

/* Create an expression to allow read access to users named guest whose
clients are in the netscape.com domain. */

expr2 = ACL_ExprNew(ACL_EXPR_TYPE_ALLOW);
rv = ACL_ExprAddArg(&err, expr2, "read");
rv = ACL_ExprTerm(&err, expr2, "user", CMP_OP_EQ, "guest");
rv = ACL_ExprTerm(&err, expr2, "dns", CMP_OP_EQ, "*.netscape.com");
rv = ACL_ExprAnd(&err, expr2);
```

```
/* Append the expression to the ACL. */
rv = ACL_ExprAppend(&err, acl, expr2);

...
```

The example above is equivalent to adding the following statements to an ACL file:

```
acl "My New ACL"

deny (write)
   user = "guest";

allow (read)
   user = "guest" and dns = "*.netscape.com";
```

See Also ACL_ExprNew(), ACL_AclNew().

ACL_ExprClearPFlags()

The `ACL_ExprClearPFlags()` function clears any processing flags for an expression. (See the documentation on ACL_ExprSetPFlags() for more information about these processing flags.)

Syntax

```
#include "aclapi.h"
NSAPI_PUBLIC
int ACL_ExprClearPFlags(NSErr_t *errp, ACLExprHandle_t *expr);
```

Parameters This function has the following parameters:

`errp`	Pointer to the header for a list of error frames.
`expr`	Pointer to the attribute expression that you want to clear the processing flags for.

Returns 0 if successful, or a negative number if an error occurred.

Note This function is part of a low-level interface that may be useful for defining ACLs without using the ACL file syntax. Because this is a low-level interface, future support of this interface is not guaranteed. Wherever possible, you should use the ACL_ParseFile() and ACL_ParseString() functions instead.

Example The following example sets and clears the processing flags for an expression.

```
#include "aclapi.h"
```

```
ACLExprHandle_t *expr;
NSErr_t err;
int rv;

...

/* Create an expression to deny write access to users named guest. */
expr = ACL_ExprNew(ACL_EXPR_TYPE_DENY);
rv = ACL_ExprAddArg(&err, expr, "write");
rv = ACL_ExprTerm(&err, expr, "user", CMP_OP_EQ, "guest");

/* Add the "absolute" flag to the expression. */
ACL_ExprSetPFlags(&err, expr, ACL_PFLAG_ABSOLUTE);

...

/* Clear all flags. */
ACL_ExprClearPFlags(&err, expr);

...
```

See Also ACL_ExprSetPFlags().

ACL_ExprDestroy()

The `ACL_ExprDestroy()` function frees an attribute expression from memory.

Syntax

```
#include "aclapi.h"
NSAPI_PUBLIC
void ACL_ExprDestroy(ACLExprHandle_t *expr);
```

Parameters This function has the following parameters:

`expr`	Pointer to the attribute expression that you want to free from memory.

Note This function is part of a low-level interface that may be useful for defining ACLs without using the ACL file syntax. Because this is a low-level interface, future support of this interface is not guaranteed. Wherever possible, you should use the `ACL_ParseFile()` and `ACL_ParseString()` functions instead.

Example The following example creates and then frees an expression from memory.

```
#include "aclapi.h"

ACLExprHandle_t *expr;
NSErr_t err;
int rv;
```

```
/* Create an expression to deny write access to users named guest. */
expr = ACL_ExprNew(ACL_EXPR_TYPE_DENY);
rv = ACL_ExprAddArg(&err, expr, "write");
rv = ACL_ExprTerm(&err, expr, "user", CMP_OP_EQ, "guest");

...

ACL_ExprDestroy(expr);

...
```

See Also ACL_ExprNew(), ACL_AclDestroy(), ACL_ListDestroy().

ACL_ExprGetDenyWith()

The ACL_ExprGetDenyWith() function gets the deny_type and deny_response values for an expression. These values are used by the server when a user attempts to access a resource without the appropriate permissions.

For example, if a user does not have read permissions to a document but attempts to read that document, the server redirects the user to the URL specified by deny_response. deny_type specifies the type of URL (partial or absolute).

This function applies only to expressions of type ACL_EXPR_TYPE_RESPONSE.

Syntax

```
#include "aclapi.h"
NSAPI_PUBLIC
int ACL_ExprGetDenyWith(NSErr_t *errp, ACLExprHandle_t *expr,
   char **deny_type, char **deny_response);
```

Parameters This function has the following parameters:

errp	Pointer to the header for a list of error frames.
expr	Pointer to the attribute expression (of type ACL_EXPR_TYPE_RESPONSE) that you want to query.

`deny_type`	Pointer to the type of URL returned to the user who failed to access the resource. This type can be one of the following: • `url` Indicates that `deny_response` specifies the absolute URL where the user is redirected. • `uri` Indicates that `deny_response` specifies the partial URL where the user is redirected.
`deny_response`	Pointer to the partial or absolute URL where the user is redirected.

Returns 0 if successful, or a negative number if an error occurred.

Note This function is part of a low-level interface that may be useful for defining ACLs without using the ACL file syntax. Because this is a low-level interface, future support of this interface is not guaranteed. Wherever possible, you should use the `ACL_ParseFile()` and `ACL_ParseString()` functions instead.

Example The following example gets the URL of the file presented to users who attempt to access a resource without the rights to that resource.

```
#include "aclapi.h"

ACLExprHandle_t *expr;
NSErr_t err;
char *deny_type;
char *deny_response;
int rv;

...

rv = ACL_ExprGetDenyWith(&err, expr, &deny_type, &deny_response);

...
```

See Also `ACL_ExprSetDenyWith()`.

ACL_ExprNew()

The `ACL_ExprNew()` function creates an attribute expression.

When you are done using this expression, you should free it from memory by calling the `ACL_ExprDestroy()` function.

Syntax

```
#include "aclapi.h"
NSAPI_PUBLIC
ACLExprHandle_t *ACL_ExprNew(const ACLExprType_t expr_type);
```

Parameters This function has the following parameters:

`expr_type` Type of attribute expression you want to create. You can specify one of the following types:

- `ACL_EXPR_TYPE_ALLOW`
  This type is equivalent to an `allow` statement in an ACL file. This expression will allow access or privileges to a resource.
- `ACL_EXPR_TYPE_DENY`
  This type is equivalent to a `deny` statement in an ACL file. This expression will deny access or privileges to a resource.
- `ACL_EXPR_TYPE_AUTH`
  This type is equivalent to an `authenticate` statement in an ACL file. This expression specifies the authentication method and the source of the authenticated information (such as a database).
- `ACL_EXPR_TYPE_RESPONSE`
  This type is equivalent to a `deny with url` statement in an ACL file. This expression specifies the URL that users will be redirected to if they do not have access or permissions to this resource.

Returns If successful, a new pointer to `ACLExprHandle_t`, which represents the new attribute expression. If unable to allocate memory for a new expression, the function returns NULL.

Note This function is part of a low-level interface that may be useful for defining ACLs without using the ACL file syntax. Because this is a low-level interface, future support of this interface is not guaranteed. Wherever possible, you should use the `ACL_ParseFile()` and `ACL_ParseString()` functions instead.

Example The following example creates a new expression.

```
#include "aclapi.h"

ACLExprHandle_t *expr;
NSErr_t err;
int rv;

/* Create an expression to deny write access to users named guest. */
```

```
expr = ACL_ExprNew(ACL_EXPR_TYPE_DENY);
rv = ACL_ExprAddArg(&err, expr, "write");
rv = ACL_ExprTerm(&err, expr, "user", CMP_OP_EQ, "guest");

...
```

See Also ACL_ExprDestroy().

ACL_ExprNot()

The `ACL_ExprNot()` function uses a NOT operator to invert the preceding terms or subexpression in an attribute expression.

This function applies only to expressions of type `ACL_EXPR_TYPE_ALLOW` or `ACL_EXPR_TYPE_DENY`.

Syntax

```
#include "aclapi.h"
NSAPI_PUBLIC
int ACL_ExprNot(NSErr_t *errp, ACLExprHandle_t *acl_expr);
```

Parameters This function has the following parameters:

`errp`	Pointer to the header for a list of error frames.
`acl_expr`	Pointer to the attribute expression (of type `ACL_EXPR_TYPE_ALLOW` or `ACL_EXPR_TYPE_DENY`) that you want to modify.

Returns 0 if successful, or a negative number if an error occurred.

Note This function is part of a low-level interface that may be useful for defining ACLs without using the ACL file syntax. Because this is a low-level interface, future support of this interface is not guaranteed. Wherever possible, you should use the ACL_ParseFile() and ACL_ParseString() functions instead.

Example The following example defines an attribute expression in which write access to a resource is denied if the subject is not a member of the `Development` group.

```
#include "aclapi.h"

ACLExprHandle_t *expr;
NSErr_t err;
int rv;

...
```

```
/* Create a new expression that "denies" access. */
expr = ACL_ExprNew(ACL_EXPR_TYPE_DENY);

/* Specify that you want to deny "write" access. */
rv = ACL_ExprAddArg(&err, expr, "write");

/* Specify that this applies to anyone who is not a member of the
Development group. */

rv = ACL_ExprTerm(&err, expr, "group", CMP_OP_EQ, "Development");
rv = ACL_ExprNot(&err, expr);

...
```

The example above is equivalent to adding the following statements to an ACL file:

```
deny (write)
   not group = "Development";
```

See Also ACL_ExprAnd(), ACL_ExprOr().ACL_ExprTerm().

ACL_ExprOr()

The `ACL_ExprOr()` function uses a logical OR operator to combine the preceding two terms or subexpressions in an attribute expression.

This function applies only to expressions of type `ACL_EXPR_TYPE_ALLOW` or `ACL_EXPR_TYPE_DENY`.

Syntax

```
#include "aclapi.h"
NSAPI_PUBLIC
int ACL_ExprOr(NSErr_t *errp, ACLExprHandle_t *acl_expr);
```

Parameters This function has the following parameters:

`errp`	Pointer to the header for a list of error frames.
`acl_expr`	Pointer to the attribute expression (of type `ACL_EXPR_TYPE_ALLOW` or `ACL_EXPR_TYPE_DENY`) that you want to modify.

Returns 0 if successful, or a negative number if an error occurred.

Note This function is part of a low-level interface that may be useful for defining ACLs without using the ACL file syntax. Because this is a low-level interface, future support of this interface is not guaranteed. Wherever possible, you should use the ACL_ParseFile() and ACL_ParseString() functions instead.

Example The following example defines an attribute expression in which write access to a resource is denied if the subject is the user guest or mozilla and if the subject's client machine is outside the domain netscape.com.

```
#include "aclapi.h"

ACLExprHandle_t *expr;
NSErr_t err;
int rv;

...

/* Create a new expression that "denies" access. */
expr = ACL_ExprNew(ACL_EXPR_TYPE_DENY);

/* Specify that you want to deny "write" access. */
rv = ACL_ExprAddArg(&err, expr, "write");

/* Specify that this applies to users named "guest" or "mozilla". */
rv = ACL_ExprTerm(&err, expr, "user", CMP_OP_EQ, "guest");
rv = ACLExprTerm(&err, expr, "user", CMP_OP_EQ, "mozilla");
rv = ACL_ExprOr(&err, expr);

/* Specify that this applies only if users are accessing the resource
from client machines outside the netscape.com domain. */

ACL_ExprTerm(&err, expr, "dns", CMP_OP_EQ, "*.netscape.com");
ACL_ExprNot(&err, expr);
ACL_ExprAnd(&err, expr);

...
```

The example above is equivalent to adding the following statements to an ACL file:

```
deny (write)
   (user = "guest" or user = "mozilla") and
   not dns = "*.netscape.com";
```

See Also ACL_ExprAnd(), ACL_ExprNot().ACL_ExprTerm().

ACL_ExprSetDenyWith()

The `ACL_ExprSetDenyWith()` function sets the `deny_type` and `deny_response` values for an expression. These values are used by the server when a user attempts to access a resource without the appropriate permissions.

For example, if a user does not have read permissions to a document but attempts to read that document, the server redirects the user to the URL specified by `deny_response`. `deny_type` specifies the type of URL (partial or absolute).

This function applies only to expressions of type `ACL_EXPR_TYPE_RESPONSE`.

Syntax

```
#include "aclapi.h"
NSAPI_PUBLIC
int ACL_ExprSetDenyWith(NSErr_t *errp, ACLExprHandle_t *expr,
   char *deny_type, char *deny_response);
```

Parameters This function has the following parameters:

`errp`	Pointer to the header for a list of error frames.
`expr`	Pointer to the attribute expression (of type `ACL_EXPR_TYPE_RESPONSE`) that you want to modify.
`deny_type`	The type of URL returned to the user who failed to access the resource. This type can be one of the following: • `url` Choose this type if `deny_response` specifies the absolute URL that the user is redirected to. • `uri` Choose this type if `deny_response` specifies a partial URL that the user is redirected to.
`deny_response`	Partial or absolute URL where the user is redirected.

Returns 0 if successful, or a negative number if an error occurred.

Note This function is part of a low-level interface that may be useful for defining ACLs without using the ACL file syntax. Because this is a low-level interface, future support of this interface is not guaranteed. Wherever possible, you should use the `ACL_ParseFile()` and `ACL_ParseString()` functions instead.

Example The following example creates a new expression to deny access to a resource. Users without access rights who attempt to access this resource will be redirected to the file `noaccess.html`.

```
#include "aclapi.h"

ACLExprHandle_t *expr;
NSErr_t err;
int rv;

...

expr = ACL_ExprNew(ACL_EXPR_TYPE_RESPONSE);
rv = ACL_ExprSetDenyWith(&err, expr, "uri", "noaccess.html");

...
```

See Also `ACL_ExprGetDenyWith()`.

ACL_ExprSetPFlags()

The `ACL_ExprSetPFlags()` function sets the processing flags for an expression. For example, you can set the ACL_PFLAG_ABSOLUTE flag, which is equivalent to the `absolute` keyword in an ACL file.

Syntax

```
#include "aclapi.h"
NSAPI_PUBLIC
int ACL_ExprSetPFlags(NSErr_t *errp, ACLExprHandle_t *expr,
   PFlags_t flags);
```

Parameters This function has the following parameters:

`errp`	Pointer to the header for a list of error frames.
`expr`	Pointer to the attribute expression that you want to modify.
`flags`	Flags that you want to set (see the list below for details).

The `flags` argument can have one of the following values:

Table 9.14 Bit flags that you can apply to access control entries

Bit Flag	Equivalent Keyword in ACL File	Description
`ACL_PFLAG_ABSOLUTE`	`absolute`	Specifies that the results of the current expression override the results of any subsequent expressions. For example, if a directory's ACL denies access and has this flag set, any resources within the directory (or its subdirectories) cannot be accessed, even if those resources have ACLs that allow access.
`ACL_PFLAG_TERMINAL`	`static`	Specifies that the ACL only applies to its associated container object itself, not to resources or container objects within this object. For example, if a directory's ACL has this flag set, this ACL is not considered when evaluating access to resources or subdirectories within the directory.
`ACL_PFLAG_CONTENT`	`content`	Specifies that the ACL applies to the contents of its associated container object, not to the container itself. For example, if a directory's ACL has this flag set, the ACL applies only to the resources and subdirectories within the directory. The ACL does not apply to the directory itself.

Returns 0 if successful, or a negative number if an error occurred.

Note This function is part of a low-level interface that may be useful for defining ACLs without using the ACL file syntax. Because this is a low-level interface, future support of this interface is not guaranteed. Wherever possible, you should use the ACL_ParseFile() and ACL_ParseString() functions instead.

Example The following example sets the processing flags for an expression.

```
#include "aclapi.h"

ACLExprHandle_t *expr;
NSErr_t err;
int rv;

...

/* Create an expression to deny write access to users named guest. */
expr = ACL_ExprNew(ACL_EXPR_TYPE_DENY);
rv = ACL_ExprAddArg(&err, expr, "write");
rv = ACL_ExprTerm(&err, expr, "user", CMP_OP_EQ, "guest");

/* Add the "absolute" flag to the expression. */
rv = ACL_ExprSetPFlags(&err, expr, ACL_PFLAG_ABSOLUTE);

...
```

The example above is equivalent to adding the following statements to an ACL file:

```
deny absolute (write)
   user = "guest";
```

See Also ACL_ExprClearPFlags().

ACL_ExprTerm()

The ACL_ExprTerm() function adds a term to the specified attribute expression. A term consists of an attribute name, a comparison operator, and a pattern that must be matched. For example:

```
   user = "mozilla";
```

The attribute name is user, the comparison operator is =, and the pattern is mozilla.

This function applies only to expressions of type ACL_EXPR_TYPE_ALLOW or ACL_EXPR_TYPE_DENY.

Syntax

```
#include "aclapi.h"
```

```
NSAPI_PUBLIC
int ACL_ExprTerm(NSErr_t *errp, ACLExprHandle_t *acl_expr,
   char *attr_name, CmpOp_t cmp, char *attr_pattern);
```

Parameters This function has the following parameters:

`errp`	Pointer to the header for a list of error frames.
`acl_expr`	Pointer to the attribute expression (of type `ACL_EXPR_TYPE_ALLOW` or `ACL_EXPR_TYPE_DENY`) to which you want to add the term.
`attr_name`	Attribute name in the term that you want to add.
`cmp`	Comparison operator in the term that you want to add. The operator can be one of the following: • `CMP_OP_EQ` Equality operator (=) • `CMP_OP_NE` Not Equals operator (!=) • `CMP_OP_GT` Greater Than operator (>) • `CMP_OP_LT` Less Than operator (<) • `CMP_OP_GE` Greater Than Or Equal To operator (>=) • `CMP_OP_LE` Less Than Or Equal To operator (<=)
`attr_pattern`	Pattern or value in the term that will be used for comparison.

Returns 0 if successful, or a negative number if an error occurred.

Note This function is part of a low-level interface that may be useful for defining ACLs without using the ACL file syntax. Because this is a low-level interface, future support of this interface is not guaranteed. Wherever possible, you should use the `ACL_ParseFile()` and `ACL_ParseString()` functions instead.

Example The following example defines an attribute expression in which write access to a resource is denied if the subject is the user `guest`.

```
#include "aclapi.h"
```

```
ACLExprHandle_t *expr;
NSErr_t err;
int rv;

...

/* Create a new expression that "denies" access. */
expr = ACL_ExprNew(ACL_EXPR_TYPE_DENY);

/* Specify that you want to deny "write" access. */
rv = ACL_ExprAddArg(&err, expr, "write");

/* Specify that this applies to users named "guest". */
rv = ACL_ExprTerm(&err, expr, "user", CMP_OP_EQ, "guest");

...
```

The example above is equivalent to adding the following statements to an ACL file:

```
deny (write)
   user = "guest";
```

See Also ACL_ExprAnd(), ACL_ExprNot().ACL_ExprOr().

ACL_FileDeleteAcl()

The ACL_FileDeleteAcl() function deletes an ACL from an ACL file.

Syntax

```
#include "aclapi.h"
NSAPI_PUBLIC
int ACL_FileDeleteAcl(NSErr_t *errp, char *filename,
   char *acl_name, int flags);
```

Parameters This function has the following parameters:

errp	Pointer to the header for a list of error frames.
filename	Name and path to the ACL file containing the ACL that you want to delete.
acl_name	Name of the ACL that you want to remove from the file.
flags	If ACL_CASE_INSENSITIVE, the function uses a case-insensitive comparison to find the ACL to be deleted. If ACL_CASE_SENSITIVE, the function uses a case-sensitive comparison to find the ACL to be deleted.

Returns 0 if successful, or a negative number if an error occurred.

Example Suppose that the ACL file named /usr/netscape/suitespot/httpacl/generated.https-mozilla.acl contains the following ACL definition:

```
acl "path=/usr/netscape/suitespot/docs/index.html";

allow (all)
   user = "mozilla";
```

The following example removes this ACL (the section that defines the ACL named path=/usr/netscape/suitespot/docs/index.html) from the ACL file.

```
#include "aclapi.h"

NSErr_t err;
char *filename =
   "/usr/netscape/suitespot/httpacl/generated.https-mozilla.acl";
char *aclname = "path=/usr/netscape/suitespot/docs/index.html";
int rv;

...

rv = ACL_FileDeleteAcl(&err, filename, aclname, ACL_CASE_INSENSITIVE);

...
```

See Also ACL_ListAclDelete(), ACL_FileGetAcl(), ACL_FileRenameAcl(), ACL_FileSetAcl().

ACL_FileGetAcl()

The ACL_FileGetAcl() function gets the text definition of an ACL from an ACL file.

Syntax

```
#include "aclapi.h"
NSAPI_PUBLIC
int ACL_FileGetAcl(NSErr_t *errp, char *filename,
   char *acl_name, char **acl_text, int flags);
```

Parameters This function has the following parameters:

errp	Pointer to the header for a list of error frames.
filename	Name and path to the ACL file containing the ACL that you want to get.
acl_name	Name of the ACL that you want to get from the file.

`acl_text`	Pointer to the buffer to hold the text definition of the ACL retrieved from the file.
`flags`	If `ACL_CASE_INSENSITIVE`, the function uses a case-insensitive comparison to find the ACL. If `ACL_CASE_SENSITIVE`, the function uses a case-sensitive comparison to find the ACL.

Returns 0 if successful, or a negative number if an error occurred.

Example Suppose that the ACL file named `/usr/netscape/suitespot/httpacl/generated.https-mozilla.acl` contains the following ACL definition:

```
acl "path=/usr/netscape/suitespot/docs/index.html";

allow (all)
   user = "mozilla";
```

The following example gets this ACL definition from the ACL file.

```
#include "aclapi.h"

NSErr_t err;
char *filename =
   "/usr/netscape/suitespot/httpacl/generated.https-mozilla.acl";
char *aclname = "path=/usr/netscape/suitespot/docs/index.html";
char *acltext;
int rv;

...

rv = ACL_FileGetAcl(&err, filename, aclname,
   &acltext, ACL_CASE_INSENSITIVE);

...
```

See Also `ACL_FileDeleteAcl()`, `ACL_FileRenameAcl()`, `ACL_FileSetAcl()`.

ACL_FileRenameAcl()

The `ACL_FileRenameAcl()` function changes the name of an ACL in an ACL file.

Syntax

```
#include "aclapi.h"
NSAPI_PUBLIC
int ACL_FileRenameAcl(NSErr_t *errp, char *filename,
   char *acl_name, char *new_acl_name, int flags);
```

Parameters This function has the following parameters:

`errp`	Pointer to the header for a list of error frames.
`filename`	Name and path to the ACL file containing the ACL that you want to rename.
`acl_name`	Current name of the ACL that you want to rename.
`new_acl_name`	New name for the ACL.
`flags`	If `ACL_CASE_INSENSITIVE`, the function uses a case-insensitive comparison to find the ACL to be renamed. If `ACL_CASE_SENSITIVE`, the function uses a case-sensitive comparison to find the ACL to be renamed.

Returns 0 if successful, or a negative number if an error occurred.

Example The following example changes the name of an ACL from `path=/usr/netscape/suitespot/docs/index.html` to `path=/usr/netscape/suitespot/docs/myfile.html`.

```
#include "aclapi.h"

NSErr_t err;
char *filename =
   "/usr/netscape/suitespot/httpacl/generated.https-mozilla.acl";
char *oldname = "path=/usr/netscape/suitespot/docs/index.html";
char *newname = "path=/usr/netscape/suitespot/docs/myfile.html";
int rv;

...

rv = ACL_FileRenameAcl(&err, filename, oldname,
   newname, ACL_CASE_INSENSITIVE);

...
```

See Also ACL_FileDeleteAcl(), ACL_FileGetAcl(), ACL_FileSetAcl().

ACL_FileSetAcl()

The `ACL_FileSetAcl()` function adds the definition of an ACL to an ACL file. If an ACL with the same name is already defined in the ACL file, this function replaces it.

Syntax

```
#include "aclapi.h"
NSAPI_PUBLIC
```

```
int ACL_FileSetAcl(NSErr_t *errp, char *filename,
   char *acl_text, int flags);
```

Parameters This function has the following parameters:

`errp`	Pointer to the header for a list of error frames.
`filename`	Name and path to the ACL file in which you want to define the ACL.
`acl_text`	Text definition of the ACL that you want to add to the ACL file. This text definition must follow the syntax rules for ACL files (for example, the text should begin with the statement `version 3.0;`). You can use `ACL_WriteString()` to generate this text. If you are using some other means to generate this text, you may want to call `ACL_ParseString()` to verify that the text complies with the ACL syntax.
`flags`	If `ACL_CASE_INSENSITIVE`, the function uses a case-insensitive comparison to find and remove any existing ACL with the same name (if an existing ACL has the same name, it is replaced with the new ACL). If `ACL_CASE_SENSITIVE`, the function uses a case-sensitive comparison to find and remove any existing ACL with the same name (if an existing ACL has the same name, it is replaced with the new ACL).

Returns 0 if successful, or a negative number if an error occurred.

Example The following example adds a definition for an ACL named `path=/usr/netscape/suitespot/docs/index.html` to an ACL file named `/usr/netscape/suitespot/httpacl/generated.https-mozilla.acl`.

```
#include "aclapi.h"

ACLListHandle_t *acl_list;
ACLHandle_t *acl;
ACLExprHandle_t *expr;
NSErr_t err;
char *filename =
   "/usr/netscape/suitespot/httpacl/generated.https-mozilla.acl";
char *aclname = "path=/usr/netscape/suitespot/docs/index.html";
char *acltext;
int rv;

...
```

```
/* Create a new access control expression to allow access. */
expr = ACL_ExprNew(ACL_EXPR_TYPE_ALLOW);

/* Allow all rights. */
rv = ACL_ExprAddArg(&err, expr, "all");

/* Give these rights to the user named mozilla. */
rv = ACL_ExprTerm(&err, expr, "user", CMP_OP_EQ, "mozilla");

/* Create a new ACL. */
acl = ACL_AclNew(&err, aclname);

/* Add the expression to the ACL. */
rv = ACL_ExprAppend(&err, acl, expr);

/* Create a new ACL list. */
acl_list = ACL_ListNew(&err);

/* Add the ACL to the list. */
rv = ACL_ListAppend(&err, acl_list, acl, 0);

/* Write the ACL list to a string. */
rv = ACL_WriteString(&err, &acltext, acl_list);

/* Use this string to add the ACL definition to the ACL file. */
rv = ACL_FileSetAcl(&err, filename, acltext, ACL_CASE_INSENSITIVE);

...
```

This example adds the following ACL definition to the ACL file:

```
acl "path=/usr/netscape/suitespot/docs/index.html";

allow (all)
   user = "mozilla";
```

See Also ACL_FileDeleteAcl(), ACL_FileGetAcl(), ACL_FileRenameAcl().

ACL_GetAttribute()

The `ACL_GetAttribute()` function gets the value of the specified attribute.

Syntax

```
#include "aclapi.h"
NSAPI_PUBLIC
int ACL_GetAttribute(NSErr_t *errp, const char *attr,
   void **val, PList_t subject, PList_t resource,
   PList_t auth_info, PList_t global_auth);
```

Parameters This function has the following parameters:

`errp`	Pointer to the header for a list of error frames.
`attr`	The attribute that you want to get.
`val`	Pointer to the buffer to hold the value of the attribute that you are getting.
`subject`	Property list for the subject.
`resource`	Property list for the resource.
`auth_info`	Authentication information for the current attribute being evaluated.
`global_auth`	Authentication information for all attributes. This argument is a two-level property list; the list contains an entry for each attribute and the authentication information for that attribute as name-value pairs.

Returns Returns one of the following values:

- `LAS_EVAL_TRUE` if the value of the attribute was successfully retrieved.
- `LAS_EVAL_FAIL` if an error occurred.
- `LAS_EVAL_INVALID` if an error occurred due to an invalid argument passed.

Description To get the value of the specified attribute, the `ACL_GetAttribute()` function does the following:

1. If the subject property list is `NULL`, returns `LAS_EVAL_FAIL`.
2. Checks the subject property list for the value of the attribute.
3. If the subject property list contains the attribute and its value, passes the value of the attribute back through the `val` argument and returns `LAS_EVAL_TRUE`.
4. If the subject property list does not contain the attribute and its value, gets the authentication method and database type information from the auth_info property list.
5. Retrieves a list of the attribute getter functions for this attribute.

6. Iterates through this list, finding and invoking the attribute getter functions registered under the same method and database type that are specified in the auth_info property list.

7. If an attribute getter function returns `LAS_EVAL_TRUE`, `ACL_GetAttribute()` verifies that the attribute has been added to the subject property list as a property:

   - If so, `ACL_GetAttribute()` returns `LAS_EVAL_TRUE` and passes the value of the attribute back as the `val` argument.

   - If, however, the attribute is not part of the subject property list, `ACL_GetAttribute()` returns `LAS_EVAL_FAIL`.

8. If an attribute getter function returns `LAS_EVAL_FAIL` or `LAS_EVAL_INVALID`, `ACL_GetAttribute()` returns with the same value.

9. If an attribute getter function returns `LAS_EVAL_DECLINE`, `ACL_GetAttribute()` repeats this process for the next matching attribute getter function in the list.

Example The following example is an evaluation function that gets the value of the attribute named `myAttr`.

```
#include "aclapi.h"

int LASMyAttrEval(NSErr_t *errp, char *attr_name, CmpOp_t comparator,
   char *attr_pattern, int *cachable, void **LAS_cookie,
   PList_t subject, PList_t resource, PList_t auth_info,
   PList_t global_auth)
{
   char *attr_val;
   int rv;

...

   rv = ACL_GetAttribute(errp, "myAttr", (void **)&attr_val, subject,
      resource, auth_info, global_auth);

   if (rv == LAS_EVAL_TRUE) {
      /* Compare attr_val to attr_pattern. */
   }

...
}
```

See Also ACL_AttrGetterRegister().

ACL_GetDefaultResult()

Calling the `ACL_GetDefaultResult()` function returns the default result of the ACL evaluation. The default result of an ACL evaluation is LAS_EVAL_FAIL. To set the default result, call the `ACL_SetDefaultResult()` function.

Syntax int ACL_GetDefaultResult(ACLEvalHandle_t *acleval)

Parameters This function has the following parameters:

`acleval`	Pointer to ACLEvalHandle_t structure that represents an ACL evaluation context.

Returns Returns the default result set for that evaluation context. The result can be one of the following values:

- ACL_RES_ALLOW
- ACL_RES_DENY
- ACL_RES_FAIL
- ACL_RES_INVALID

See Also `ACL_SetDefaultResult()`.

ACL_LasRegister()

The `ACL_LasRegister()` function registers a new Loadable Authentication Service (LAS). An LAS is a module responsible for evaluating attribute expression terms involving one or more specified attribute names.

`ACL_LasRegister()` registers a function for evaluating attribute expressions for a given attribute. The evaluation function must be of the type `LASEvalFunc_t`.

`ACL_LasRegister()` also registers a function for flushing cached data from evaluating attribute expressions. The flush function must be of the type `LASFlushFunc_t`.

Syntax

```
#include "aclapi.h"
NSAPI_PUBLIC
int ACL_LasRegister(NSErr_t *errp, char *attr_name,
   LASEvalFunc_t eval_func, LASFlushFunc_t flush_func);
```

Parameters This function has the following parameters:

`errp`	Pointer to the header for a list of error frames.
`attr_name`	Name of the attribute that this LAS is responsible for evaluating.
`eval_func`	Function to call to evaluate the attribute.
`flush_func`	Function to call to flush the cache, if attribute values are cached.

Returns 0 if successful, or a negative number if an error occurred.

Example See "LAS Code for the Email Attribute" and "LAS Code for the HTTP Referer Attribute".

See Also `LASEvalFunc_t`, `LASFlushFunc_t`.

ACL_LDAPDatabaseHandle()

The `ACL_LDAPDatabaseHandle()` function finds the internal structure representing an authentication database. If it is an LDAP database, returns the connection handle to the LDAP Server (`LDAP **ld`) and binds to the LDAP server.

You can use the connection handle in subsequent calls, if you want to perform LDAP operations on the server. Note that you can cache the `ld` and `basedn` values instead of getting them each time.

If your LDAP request returns the value `LDAP_SERVER_DOWN`, you must get the `ld` connection handle again. If you do not need to get the `basedn` again, you can pass a NULL for the `basedn` argument.

Syntax

```
#include "aclapi.h"
NSAPI_PUBLIC
int ACL_LDAPDatabaseHandle (NSErr_t *errp, const char *dbname,
   LDAP **ld, char **basedn);
```

Parameters This function has the following parameters:

`errp`	Pointer to the header for a list of error frames.
`dbname`	Name of the LDAP database. If this argument is NULL, the default database is used.
`ld`	Pointer to the connection handle to the LDAP server. If you are performing LDAP operations through LDAP API function calls, you need to pass this handle as an argument to the functions you call.
`basedn`	Pointer to the distinguished name (DN) of the root object in the LDAP directory (for example, o=Ace Industry,c=US). When you are done working with this value, free the value by calling the PERM_FREE function.

Returns `LAS_EVAL_TRUE` if successful, or `LAS_EVAL_FAIL` if an error occurs.

Example The following example gets the LDAP connection handle for the default database.

```
#include "aclapi.h"
#include "ldapapi.h"

LDAP *ld;
char *basedn;
NSErr_t err;
int rv;

...

/* Get the connection handle for the default database. */
rv = ACL_LDAPDatabaseHandle (&err, NULL, &ld, &basedn);

...
```

See Also `ACL_DatabaseSetDefault()`.

ACL_ListAclDelete()

The `ACL_ListAclDelete()` function removes a specified ACL from an `ACL list`.

Syntax

```
#include "aclapi.h"
NSAPI_PUBLIC
int ACL_AclDelete(NSErr_t *errp, ACLListHandle_t *acl_list,
```

```
  char *acl_name, int flags);
```

Parameters This function has the following parameters

`errp`	Pointer to the header for a list of error frames.
`acl_list`	Pointer to the ACL list that you want to remove the ACL from.
`acl_name`	Name of the ACL that you want to remove from the list.
`flags`	If `ACL_CASE_INSENSITIVE`, the function uses a case-insensitive comparison to find the ACL with a matching name. If `ACL_CASE_SENSITIVE`, the function uses a case-sensitive comparison to find the ACL with a matching name.

Returns 0 if successful, or a negative value if unsuccessful.

Example The following example reads in an ACL list from the ACL file for Netscape web servers. After creating the ACL list, the example finds and removes the ACL named `path=/usr/netscape/suitespot/doc/index.html`.

```
#include "aclapi.h"

ACLListHandle_t *acl_list;
NSErr_t err;
char *filename = "/usr/netscape/httpacl/generated.https-bongo.acl";
int rv;

...

acl_list = ACL_ParseFile(&err, filename);

if (acl_list) {
   rv = ACL_ListAclDelete(&err, acl_list,
      "path=/usr/netscape/suitespot/doc/index.html",
      ACL_CASE_INSENSITIVE);
}

...
```

See Also `ACL_ListAppend()`, `ACL_ListFind()`, `ACL_FileDeleteAcl()`.

ACL_ListAppend()

The `ACL_ListAppend()` function appends an ACL to an ACL list.

Syntax

```
#include "aclapi.h"
NSAPI_PUBLIC
int ACL_ListAppend(NSErr_t *errp, ACLListHandle_t *acl_list,
   ACLHandle_t *acl, int flags);
```

Parameters This function has the following parameters:

`errp`	Pointer to the header for a list of error frames.
`acl_list`	Pointer to the ACL list that you want to add this ACL to.
`acl`	Pointer to the ACL that you want to add to the list.
`flags`	Not currently used for this function.

Returns If successful, returns the number of ACLs in the list after the ACL has been appended. If not successful, returns a negative number.

Example The following example appends a newly created ACL to an ACL list.

```
#include "aclapi.h"

ACLListHandle_t *acl_list = NULL;
ACLHandle_t *acl;
ACLExprHandle_t *expr;
NSErr_t err;
int rv;

...

/* Create a new access control expression to allow access. */
expr = ACL_ExprNew(ACL_EXPR_TYPE_ALLOW);

/* Allow all rights. */
rv = ACL_ExprAddArg(&err, expr, "all");

/* Give these rights to the user named mozilla. */
rv = ACL_ExprTerm(&err, expr, "user", CMP_OP_EQ, "mozilla");

/* Create a new ACL. */
acl = ACL_AclNew(&err, "path=/usr/netscape/suitespot/docs/index.html");

/* Add this expression to the ACL. */
rv = ACL_ExprAppend(&err, acl, expr);

/* Create a new ACL list. */
acl_list = ACL_ListNew(&err);

/* Add the ACL to the list. */
rv = ACL_ListAppend(&err, acl_list, acl, 0);

...
```

See Also ACL_ListConcat(), ACL_ListDestroy(), ACL_ListFind(), ACL_ListGetNameList(), ACL_ListNew().

ACL_ListConcat()

The ACL_ListConcat() function concatenates one ACL list to another.

Syntax

```
#include "aclapi.h"
NSAPI_PUBLIC
int ACL_ListConcat(NSErr_t *errp, ACLListHandle_t *acl_list1,
   ACLListHandle_t *acl_list2, int flags);
```

Parameters This function has the following parameters:

errp	Pointer to the header for a list of error frames.
acl_list1	Pointer to the first ACL list.
acl_list2	Pointer to the second ACL list that you want concatenated to the first ACL list.
flags	Not currently used for this function.

Returns If successful, returns the number of ACLs in the ACL list after concatenation. If not successful, returns a negative number.

Example The following example reads in two ACL files and concatenates them together.

```
#include "aclapi.h"

ACLListHandle_t *acl_list, *acl_list2;
NSErr_t err;
char *filename =
   "/usr/netscape/suitespot/httpacl/generated.https-mozilla.acl";
char *filename2 = "/usr/netscape/suitespot/httpacl/my.acl";
int rv;

...

/* Read the ACL files into memory. */
acl_list = ACL_ParseFile(&err, filename);
acl_list2 = ACL_ParseFile(&err, filename2);

/* Concatenate the ACL lists. */
rv = ACL_ListConcat(&err, acl_list, acl_list2, 0);

...
```

See Also ACL_ListAppend(), ACL_ListDestroy(), ACL_ListFind(), ACL_ListGetNameList(), ACL_ListNew().

ACL_ListDestroy()

The ACL_ListDestroy() function frees an ACL list from memory.

Syntax

```
#include "aclapi.h"
NSAPI_PUBLIC
void ACL_ListDestroy(NSErr_t *errp, ACLListHandle_t *acl_list);
```

Parameters This function has the following parameters:

errp	Pointer to the header for a list of error frames.
acl_list	Pointer to the ACL list that you want to free from memory.

Example The following example frees an ACL list from memory.

```
#include "aclapi.h"

ACLListHandle_t *acl_list = NULL;
NSErr_t err;

...

/* Create a new ACL list. */
acl_list = ACL_ListNew(&err);

...

/* Work with the ACL list. */

...

/* Free the ACL list when done. */
ACL_ListDestroy(&err, acl_list);

...
```

See Also ACL_ListAppend(), ACL_ListConcat(), ACL_ListFind(), ACL_ListGetNameList(), ACL_ListNew().

ACL_ListFind()

The `ACL_ListFind()` function finds an ACL in an ACL list.

Syntax

```
#include "aclapi.h"
NSAPI_PUBLIC
ACLHandle_t *ACL_ListFind (NSErr_t *errp,
   ACLListHandle_t *acllist, char *aclname, int flags);
```

Parameters This function has the following parameters:

`errp`	Pointer to the header for a list of error frames.
`acllist`	Pointer to the ACL list that you want to search for the ACL.
`aclname`	Name of the ACL that you want to find.
`flags`	If `ACL_CASE_INSENSITIVE`, the function uses a case-insensitive comparison to find the ACL with a matching name. If `ACL_CASE_SENSITIVE`, the function uses a case-sensitive comparison to find the ACL with a matching name.

Returns If successful, returns the `ACLHandle_t` pointer representing the ACL that you searched for. If unsuccessful, returns NULL.

Example The following example finds the ACL named `path=/usr/netscape/suitespot/docs/index.html` in an ACL file loaded into memory.

```
#include "aclapi.h"

ACLListHandle_t *acl_list;
ACLHandle_t *acl;
NSErr_t err;
char *filename =
   "/usr/netscape/suitespot/httpacl/generated.https-mozilla.acl";
char *acl_name = "path=/usr/netscape/suitespot/docs/index.html";

...

/* Read the ACL file into memory. */
acl_list = ACL_ParseFile(&err, filename);

/* Find the ACL. */
acl = ACL_ListFind(&err, acl_list, acl_name, ACL_CASE_INSENSITIVE);

...
```

See Also ACL_ListAppend(), ACL_ListConcat(), ACL_ListDestroy(), ACL_ListGetNameList(), ACL_ListNew().

ACL_ListGetFirst()

The ACL_ListGetFirst() function starts the enumeration of an ACLListHandle_t structure. The function returns an ACLHandle_t pointer for the first ACL on the list and initializes the handle of an ACLListEnum_t structure, which is used to track the current position in the enumeration.

Typically, you might use this function in conjunction with the ACL_ListGetNext() function in a loop such as the following:

```
ACLListHandle_t *acl_list;
ACLHandle_t *cur_acl;
ACLListEnum_t acl_enum;

...

for (cur_acl = ACL_ListGetFirst(acl_list, &acl_enum);
   cur_acl != NULL;
   cur_acl = ACL_ListGetNext(acl_list, &acl_enum)) {

   ...

}

...
```

While you are enumerating the ACLListHandle_t structure, make sure that no ACLs are added to or removed from the ACL list.

Syntax

```
#include "aclapi.h"
NSAPI_PUBLIC
ACLHandle_t *ACL_ListGetFirst(ACLListHandle_t *acl_list,
   ACLListEnum_t *acl_enum);
```

Parameters This function has the following parameters:

acl_list	Pointer to the ACL list.
acl_enum	Pointer to an uninitialized enumeration handle. Use this handle to keep track of the current position in the enumeration. As you call the ACL_ListGetNext() function to get subsequent ACLs in the list, the enumeration handle is updated to point to the current position in the list.

Returns If successful, returns the `ACLHandle_t` pointer representing the first ACL in the list. If unsuccessful, returns NULL.

Example The following example reads and parses an ACL file into memory and then enumerates the ACLs in the file.

```
#include "aclapi.h"

ACLListHandle_t *acl_list;
ACLHandle_t *cur_acl;
ACLListEnum_t acl_enum;
NSErr_t err;
char *filename =
   "/usr/netscape/suitespot/httpacl/generated.https-mozilla.acl";

...

/* Read the ACL file into memory. */
acl_list = ACL_ParseFile(&err, filename);

for (cur_acl = ACL_ListGetFirst(acl_list, &acl_enum);
   cur_acl != NULL;
   cur_acl = ACL_ListGetNext(acl_list, &acl_enum)) {

   /* Process each ACL. */
   ...

}

...
```

See Also ACL_ListGetNext(), ACL_ListFind().

ACL_ListGetNameList()

The `ACL_ListGetNameList()` function gets the names of the ACLs in an ACL list.

When you are done working with this list of names, free the list from memory by calling the ACL_NameListDestroy() function.

Syntax

```
#include "aclapi.h"
NSAPI_PUBLIC
int ACL_ListGetNameList(NSErr_t *errp,
   ACLListHandle_t *acl_list, char ***name_list);
```

Parameters This function has the following parameters:

`errp`	Pointer to the header for a list of error frames.
`acl_list`	Pointer to the ACL list.
`name_list`	Pointer to the list of the names of the ACLs that are in the ACL list specified by `acl_list`.

Returns 0 if successful, or a negative number if an error occurred.

Example The following example gets the names of the ACLs in an ACL file.

```
#include "aclapi.h"

ACLListHandle_t *acl_list;
NSErr_t err;
char *filename =
   "/usr/netscape/suitespot/httpacl/generated.https-myhost.acl";
int rv;
char **name_list;

...

/* Read the ACL file into memory. */
acl_list = ACL_ParseFile(&err, filename);

/* Get a list of the names of the ACLs. */
rv = ACL_ListGetNameList(&err, acl_list, &name_list);

...
```

See Also ACL_NameListDestroy(), ACL_ListAppend(), ACL_ListConcat(), ACL_ListDestroy(), ACL_ListFind(), ACL_ListNew().

ACL_ListGetNext()

The `ACL_ListGetNext()` function gets the next ACL in the enumeration of an `ACLListHandle_t` structure. You start the enumeration by calling the `ACL_ListGetFirst()` function.

For more information, see "ACL_ListGetFirst()".

Syntax

```
#include "aclapi.h"
NSAPI_PUBLIC
ACLHandle_t *ACL_ListGetNext(ACLListHandle_t *acl_list,
   ACLListEnum_t *acl_enum);
```

Parameters This function has the following parameters:

`acl_list`	Pointer to the ACL list.
`acl_enum`	Pointer to an enumeration handle initialized by the `ACL_ListGetFirst()` function call. Use this handle to keep track of the current position in the enumeration. As you call the `ACL_ListGetNext()` function to get subsequent ACLs in the list, the enumeration handle is updated to point to the current position in the list.

Returns If successful, returns the `ACLHandle_t` representing the specified ACL. If unsuccessful, returns NULL.

Example See the example under "ACL_ListGetFirst()".

See Also `ACL_ListGetFirst()`, `ACL_ListFind()`.

ACL_ListNew()

The `ACL_ListNew()` function creates a new ACL list. Initially, this list is empty, but you can add ACLs to the list by calling the `ACL_ListAppend()` function.

When you are done using this ACL list, free it from memory by calling the `ACL_ListDestroy()` function.

Syntax

```
#include "aclapi.h"
NSAPI_PUBLIC
ACLListHandle_t *ACL_ListNew(NSErr_t *errp);
```

Parameters This function has the following parameters:

`errp`	Pointer to the header for a list of error frames.

Returns If successful, a new `ACLListHandle_t` pointer, which represents the new ACL list. If unsuccessful, the function returns NULL.

Example The following example creates a new ACL list.

```
#include "aclapi.h"

ACLListHandle_t *acl_list;
ACLHandle_t *acl;
NSErr_t err;
int rv;
```

```
...

/* Create a new ACL list. */
acl_list = ACL_ListNew(&err);

/* Add the ACL to the list. */
rv = ACL_ListAppend(&err, acl_list, acl, 0);

...
```

See Also ACL_ListAppend(), ACL_ListConcat(), ACL_ListDestroy(), ACL_ListFind(), ACL_ListGetNameList().

ACL_MethodFind()

The `ACL_MethodFind()` function finds the method type with the specified name.

Syntax

```
#include "aclapi.h"
NSAPI_PUBLIC
int ACL_MethodFind(NSErr_t *errp, const char *name,
   ACLMethod_t *t);
```

Parameters This function has the following parameters:

`errp`	Pointer to the header for a list of error frames.
`name`	Name of the method that you want to find.
`t`	Pointer to a buffer to contain method information.

Returns 0 if successful, or a negative number if an error occurred.

See Also ACL_AuthInfoGetMethod(), ACL_MethodGetDefault(), ACL_MethodIsEqual(), ACL_MethodNameIsEqual(), ACL_MethodRegister(), ACL_MethodSetDefault().

ACL_MethodGetDefault()

The `ACL_MethodGetDefault()` function returns the default method.

Syntax

```
#include "aclapi.h"
NSAPI_PUBLIC
ACLMethod_t ACL_MethodGetDefault(NSErr_t *errp);
```

Parameters This function has the following parameters:

`errp`	Pointer to the header for a list of error frames.

Returns The default method.

See Also `ACL_AuthInfoGetMethod()`, `ACL_MethodFind()`, `ACL_MethodIsEqual()`, `ACL_MethodNameIsEqual()`, `ACL_MethodRegister()`, `ACL_MethodSetDefault()`.

ACL_MethodIsEqual()

The `ACL_MethodIsEqual()` function determines whether two methods are equal.

Syntax

```
#include "aclapi.h"
NSAPI_PUBLIC
int ACL_MethodIsEqual(NSErr_t *errp, const ACLMethod_t t1,
   const ACLMethod_t t2);
```

Parameters This function has the following parameters:

`errp`	Pointer to the header for a list of error frames.
`t1`	One of the two methods that you want to compare.
`t2`	The other method that you want to compare.

Returns 1 if the methods are equal, 0 if not.

See Also `ACL_AuthInfoGetMethod()`, `ACL_MethodFind()`, `ACL_MethodGetDefault()`, `ACL_MethodNameIsEqual()`, `ACL_MethodRegister()`, `ACL_MethodSetDefault()`.

ACL_MethodNameIsEqual()

The `ACL_MethodNameIsEqual()` function determines if a given method has the specified name.

Syntax

```
#include "aclapi.h"
NSAPI_PUBLIC
```

```
int ACL_MethodNameIsEqual(NSErr_t *errp, const ACLMethod_t t,
   const char *name);
```

Parameters This function has the following parameters:

`errp`	Pointer to the header for a list of error frames.
`t`	Method that you want to compare against the method name.
`name`	Name of a method that you want to compare against the method.

Returns 1 if the method has the given name, 0 if not.

See Also `ACL_AuthInfoGetMethod()`, `ACL_MethodFind()`, `ACL_MethodGetDefault()`, `ACL_MethodNameIsEqual()`, `ACL_MethodRegister()`, `ACL_MethodSetDefault()`.

ACL_MethodRegister()

The `ACL_MethodRegister()` function registers a method.

Syntax

```
#include "aclapi.h"
NSAPI_PUBLIC
int ACL_MethodRegister(NSErr_t *errp, const char *name,
   ACLMethod_t *t);
```

Parameters This function has the following parameters:

`errp`	Pointer to the header for a list of error frames.
`name`	Method name string. You can free this from memory after the function returns.
`t`	Pointer to the buffer to return the method.

Returns 0 if successful, or a negative number if an error occurred.

See Also `ACL_AuthInfoGetMethod()`, `ACL_MethodFind()`, `ACL_MethodGetDefault()`, `ACL_MethodIsEqual()`, `ACL_MethodNameIsEqual()`, `ACL_MethodSetDefault()`.

ACL_MethodSetDefault()

The `ACL_MethodSetDefault()` function specifies the method used as the default method.

Syntax

```
#include "aclapi.h"
NSAPI_PUBLIC
int ACL_MethodSetDefault(NSErr_t *errp, const ACLMethod_t t);
```

Parameters This function has the following parameters:

`errp`	Pointer to the header for a list of error frames.
`t`	Method that you want to set as the default method.

Returns 0 if successful.

See Also `ACL_AuthInfoGetMethod()`, `ACL_MethodFind()`, `ACL_MethodGetDefault()`, `ACL_MethodIsEqual()`, `ACL_MethodNameIsEqual()`, `ACL_MethodRegister()`.

ACL_ModuleRegister()

The `ACL_ModuleRegister()` function registers an LAS module and runs the initialization function associated with that module. The initialization function must be a function of the type `AclModuleInitFunc`.

The equivalent built-in is `acl-register-module`. You can also register an LAS module by including this function in an `Init` directive in your `obj.conf` file. For example, the following directives initialize the `lasemail` module:

```
Init fn="load-modules" funcs="LASEmailModuleInit" shlib="lasemail.so"
Init fn=acl-register-module module="lasemail" func="LASEmailModuleInit"
```

Syntax

```
#include "aclapi.h"
NSAPI_PUBLIC
int ACL_ModuleRegister(NSErr_t *errp, const char *module_name,
   AclModuleInitFunc func);
```

Parameters This function has the following parameters:

`errp`	Pointer to the header for a list of error frames.
`module_name`	Name of the module that you want to register. Provide a name such as the simple name of the DLL or `.so` file, for example, `lasemail`. This name is used in error messages.
`func`	Initialization function to be executed when the module is registered.

Returns 0 if successful, or a negative number if an error occurred.

See Also `AclModuleInitFunc`, `ACL_LasRegister()`.

ACL_NameListDestroy()

The `ACL_NameListDestroy()` function frees a list of ACL names from memory. Call this function to free the list of names created from a `ACL_ListGetNameList()` function call.

Syntax

```
#include "aclapi.h"
NSAPI_PUBLIC
int ACL_NameListDestroy(NSErr_t *errp, char **name_list);
```

Parameters This function has the following parameters:

`errp`	Pointer to the header for a list of error frames.
`name_list`	List of ACL names that you want to free from memory.

Returns 0 if successful, or a negative number if an error occurred.

Example The following example frees a list of names after they are no longer used.

```
#include "aclapi.h"

ACLListHandle_t *acl_list;
NSErr_t err;
char *filename =
   "/usr/netscape/suitespot/httpacl/generated.https-myhost.acl";
char **name_list;
int rv;

...

/* Read the ACL file into memory. */
acl_list = ACL_ParseFile(&err, filename);
```

```
/* Get a list of the names of the ACLs. */
rv = ACL_ListGetNameList(&err, acl_list, &name_list);

...

/* Work with the list of names. */

...

/* Free the list of names. */
rv = ACL_NameListDestroy(&err, name_list);

...
```

See Also ACL_ListGetNameList().

ACL_ParseFile()

The `ACL_ParseFile()` function reads an ACL file and generates an ACL list from the contents of that file.

Syntax

```
#include "aclapi.h"
NSAPI_PUBLIC
ACLListHandle_t *ACL_ParseFile(NSErr_t *errp, char *filename);
```

Parameters This function has the following parameters:

`errp`	Pointer to the header for a list of error frames.
`filename`	Name of the ACL file to read.

Returns If successful, returns an `ACLListHandle_t` pointer representing the ACL list read from the file. If unsuccessful, returns NULL.

Example The following example reads an ACL file into memory.

```
#include "aclapi.h"

ACLListHandle_t *acl_list;
NSErr_t err;
char *filename =
   "/usr/netscape/suitespot/httpacl/generated.https-mozilla.acl";

...

/* Read the ACL file into memory. */
acl_list = ACL_ParseFile(&err, filename);

...
```

See Also ACL_ParseString(), ACL_WriteFile(), ACL_WriteString().

ACL_ParseString()

The ACL_ParseString() function reads a string of ACL information and generates an ACL list from that string.

Syntax

```
#include "aclapi.h"
NSAPI_PUBLIC
ACLListHandle_t *ACL_ParseString(NSErr_t *errp, char *buffer);
```

Parameters This function has the following parameters:

errp	Pointer to the header for a list of error frames.
buffer	String containing the ACL information to be read.

Returns If successful, returns an ACLListHandle_t pointer representing the ACL list read from the string. If unsuccessful, returns NULL.

Example The following example verifies that the text definition of an ACL list can be parsed correctly before saving the definition to an ACL file.

```
#include "aclapi.h"

ACLListHandle_t *acl_list;
NSErr_t err;
char *filename =
   "/usr/netscape/suitespot/httpacl/generated.https-mozilla.acl";
char *aclname = "path=/usr/netscape/suitespot/docs/index.html";
char *acltext;
int rv;

...

/* Parse a string containing the text definition of an ACL list. */
acl_list = ACL_ParseString(&err, acltext);

/* If the string parses correctly, add it to the ACL file. */
if (acl_list != NULL) {
   rv = ACL_FileSetAcl(&err, filename, acltext, 0);
}

...
```

See Also ACL_ParseFile(), ACL_WriteFile(), ACL_WriteString().

ACL_SetDefaultResult()

Normally, the default result of the ACL evaluation is LAS_EVAL_FAIL. Use the `ACL_SetDefaultResult()` function to change the default result to LAS_EVAL_TRUE.

Syntax int ACL_SetDefaultResult(NSErr_t *errp, ACLEvalHandle_t *acleval, int result)

Parameters This function has the following parameters:

`errp` Pointer to the header for a list of error frames.

`acleval` Pointer to the ACL_EvalHandle_t structure that represents an ACl evaluation context.

`result` the default result that you want to set for that evaluation context. The result can be one of the following values:

- ACL_RES_ALLOW
- ACL_RES_DENY
- ACL_RES_FAIL
- ACL_RES_INVALID

Returns Returns 0 if successful or -1 if an error occurred (for example, if you specified an invalid value for result).

See Also `ACL_GetDefaultResult()`.

ACL_SetupEval()

The `ACL_SetupEval()` function checks the access allowed to a given resource.

Note that this function is declared in `nsapi.h`, not in `aclapi.h`.

Syntax

```
#include "nsapi.h"
NSAPI_PUBLIC
int ACL_SetupEval(ACLListHandle_t *acllist, Session *sn,
   Request *rq, char **rights, char **map_generic,
```

```
   const char *user)
```

Parameters This function has the following parameters:

`acllist`	Pointer to the ACL list
`sn`	Session structure pointer in NSAPI
`rq`	Request structure pointer in NSAPI
`rights`	Array of the rights that you want to check.
`map_generic`	Mapping of generic access rights to specific access rights
`user`	User requesting access to the resource.

Returns .`ACL_RES_ALLOW`, if access is allowed, or `ACL_RES_DENY`, if access is denied.

ACL_WriteFile()

The `ACL_WriteFile()` function writes an ACL list in memory to a text file.

Syntax

```
#include "aclapi.h"
NSAPI_PUBLIC
int ACL_WriteFile(NSErr_t *errp, char *filename,
   ACLListHandle_t *acllist);
```

Parameters This function has the following parameters:

`errp`	Pointer to the header for a list of error frames.
`filename`	Name of the ACL file in which to save the ACL list.
`acllist`	Pointer to the ACL list that you want written to the file.

Returns 0 if successful, or an error code if an error occurs (for example, the function might return `ACLERROPEN` or `ACLERRNOMEM` if an error occurs).

Example The following example creates an ACL file that contains a simple ACL list.

```
#include "aclapi.h"

ACLListHandle_t *acl_list;
ACLHandle_t *acl;
ACLExprHandle_t *expr;
NSErr_t err;
char *filename =
```

```
   "/usr/netscape/suitespot/httpacl/generated.https-mozilla.acl";
char *aclname = "path=/usr/netscape/suitespot/docs/index.html";
int rv;

...

/* Create a new access control expression to allow access. */
expr = ACL_ExprNew(ACL_EXPR_TYPE_ALLOW);

/* Allow all rights. */
rv = ACL_ExprAddArg(&err, expr, "all");

/* Give these rights to the user named mozilla. */
rv = ACL_ExprTerm(&err, expr, "user", CMP_OP_EQ, "mozilla");

/* Create a new ACL. */
acl = ACL_AclNew(&err, aclname);

/* Add this expression to the ACL. */
rv = ACL_ExprAppend(&err, acl, expr);

/* Create a new ACL list. */
acl_list = ACL_ListNew(&err);

/* Add the ACL to the list. */
rv = ACL_ListAppend(&err, acl_list, acl, 0);

/* Write the ACL list to a file. */
rv = ACL_WriteFile(&err, filename, acl_list);

...
```

See Also ACL_ParseFile(), ACL_ParseString(), ACL_WriteString().

ACL_WriteString()

The `ACL_WriteString()` function writes an ACL list to a text string.

Syntax

```
#include "aclapi.h"
NSAPI_PUBLIC
int ACL_WriteString(NSErr_t *errp, char **acl,
   ACLListHandle_t *acllist);
```

Parameters This function has the following parameters:

`errp`	Pointer to the header for a list of error frames.
`acl`	Pointer to the text string in which to write the ACL list.
`acllist`	Pointer to the ACL list that you want written to the text string.

Returns 0 if successful, or a negative number if an error occurs.

Example The following example writes the text definition of an ACL list to a string.

```
#include "aclapi.h"

ACLListHandle_t *acl_list;
ACLHandle_t *acl;
ACLExprHandle_t *expr;
NSErr_t err;
char *filename =
   "/usr/netscape/suitespot/httpacl/generated.https-mozilla.acl";
char *aclname = "path=/usr/netscape/suitespot/docs/index.html";
char *acltext;
int rv;

...

/* Create a new access control expression to allow access. */
expr = ACL_ExprNew(ACL_EXPR_TYPE_ALLOW);

/* Allow all rights. */
rv = ACL_ExprAddArg(&err, expr, "all");

/* Give these rights to the user named mozilla. */
rv = ACL_ExprTerm(&err, expr, "user", CMP_OP_EQ, "mozilla");

/* Create a new ACL. */
acl = ACL_AclNew(&err, aclname);

/* Add this expression to the ACL. */
rv = ACL_ExprAppend(&err, acl, expr);

/* Create a new ACL list. */
acl_list = ACL_ListNew(&err);

/* Add the ACL to the list. */
rv = ACL_ListAppend(&err, acl_list, acl, 0);

/* Write the ACL list to a string. */
rv = ACL_WriteString(&err, &acltext, acl_list);
```

See Also ACL_ParseFile(), ACL_ParseString(), ACL_WriteFile().

DbParseFn_t

DbParseFn_t is the type definition for a function. This type describes a kind of callback function that parses a reference to an authentication database of a particular database type. You associate this function with a database type by calling the ACL_DbTypeRegister() function.

Whenever a database is registered (by calling the `ACL_DatabaseRegister()` function), the parsing function for that type of database is called.

Syntax

```
#include "aclapi.h"
NSAPI_PUBLIC
typedef int (*DbParseFn_t)(NSErr_t *errp, ACLDbType_t dbtype,
   const char *name, const char *url, PList_t plist,
   void **db);
```

Parameters Functions of this type have the following parameters:

`errp`	Pointer to the header for a list of error frames.
`dbtype`	Type of database that uses this parsing function.
`name`	Name of the database for which you want the reference parsed.
`url`	URL reference to parse into a structure, which is returned as the `db` argument.
`plist`	Property list
`db`	Pointer to parsed information from the URL reference.

Returns 0 if successful, or a negative value if unsuccessful.

See Also `ACL_DbTypeRegister()`, `ACL_DatabaseRegister()`.

LASEvalFunc_t

`LASEvalFunc_t` is the type definition for a function. This type describes a kind of callback function that is called to evaluate an attribute value expression in an ACL statement. You associate this function with the attribute for an LAS by calling the `ACL_LasRegister()` function.

For example, suppose an ACL statement contains the following lines:

```
allow (read)
   email="mozilla@netscape.com";
```

You need to define a function of the type `LASEvalFunc_t` that evaluates the `email` attribute.

Syntax

```
#include "aclapi.h"
NSAPI_PUBLIC
```

```
typedef int (*LASEvalFunc_t)(NSErr_t *errp, char *attr_name,
   CmpOp_t comparator, char *attr_pattern,
   ACLCachable_t *cachable, void **cookie, PList_t subject,
   PList_t resource, PList_t auth_info, PList_t global_auth);
```

Parameters Functions of this type have the following parameters:

`errp`	Pointer to the header for a list of error frames.
`attr_name`	Name of the attribute in the expression.
`comparator`	Comparison operator that appears in the attribute expression being evaluated (for example, `CMP_OP_EQ` if the attribute expression is `email="mozilla@netscape.com"`).
`attr_pattern`	Pattern that the attribute is compared against in the expression (for example, `"mozilla@netscape.com"`).
`cachable`	Specifies whether the evaluation results can be cached (`ACL_NOT_CACHABLE` or `ACL_INDEF_CACHABLE`).
`cookie`	If the results are cached, the pointer to the cached data.
`subject`	Property list for the subject.
`resource`	Property list for the resource.
`auth_info`	Property list containing authentication information for the current attribute.
`global_auth`	Property list containing authentication information for all attributes.

Returns Returns one of the following values:

- `LAS_EVAL_TRUE` if the evaluation completed successfully and the result was true.
- `LAS_EVAL_FALSE` if the evaluation completed successfully and the result was false.
- `LAS_EVAL_FAIL` if an error occurred.
- `LAST_EVAL_INVALID` if a configuration error occurred (for example, if an invalid argument was passed to the evaluation function).

See Also `ACL_LasRegister()`.

LASFlushFunc_t

`LASFlushFunc_t` is the type definition for a function. This type describes a kind of callback function that is called when a previously cached LAS cookie is being flushed from the ACL cache. You associate this function with the attribute for an LAS by calling the `ACL_LasRegister()` function.

Syntax

```
#include "aclapi.h"
NSAPI_PUBLIC
typedef void (*LASFlushFunc_t)(void **cookie);
```

Parameters Functions of this type have the following parameters:

`cookie`	Pointer to the cached cookie that you want flushed from the cache.

See Also `ACL_LasRegister()`.

nserrDispose()

The `nserrDispose()` function frees a list of error frames from memory.

This function does not free the header for the list of error frames, because the header is typically not dynamically allocated.

Syntax

```
#include "aclapi.h"
NSAPI_PUBLIC
void nserrDispose(NSErr_t *errp);
```

Parameters This function has the following parameters:

`errp`	Pointer to the header for a list of error frames that you want to free.

Example The following example frees a list of error frames.

```
#include "aclapi.h"

NSErr_t err;

...

/* Free this list of frames */
nserrDispose(&err);
```

```
...
```

See Also nserrFAlloc(), nserrFFree(), nserrGenerate().

nserrFAlloc()

The nserrFAlloc() function allocates memory for a new error frame structure. (nserrGenerate() calls this function when generating a new error frame.)

To allocate the memory, this function either calls the allocator function specified by the err_falloc field in errp or MALLOC (if errp is NULL or if the err_falloc field does not specify any allocator function).

Syntax

```
#include "aclapi.h"
NSAPI_PUBLIC
NSEFrame_t *nserrFAlloc(NSErr_t *errp);
```

Parameters This function has the following parameters:

errp	Pointer to the header for a list of error frames. This argument can be NULL.

Returns If successful, returns a pointer to a new error frame. (Note that this frame is not added to the list of error frames specified by the errp argument.) If unsuccessful, returns NULL.

Example The following example allocates memory for a new error frame.

```
#include "aclapi.h"

...

NSEFrame_t *efp;
NSErr_t err;

...

/* Allocate the error frame */
efp = nserrFAlloc(&err);

...
```

See Also nserrFFree(), nserrGenerate(), nserrDispose().

nserrFFree()

The `nserrFFree()` function frees an error frame structure from memory.

Before freeing the structure from memory, this function determines if the specified error frame is in the list of error frames. If so, the function removes the error frame from the list.

To free the structure, this function either calls the function specified by the `err_ffree` field in `errp` or `FREE` (if `errp` is NULL or if the `err_ffree` field does not specify any function).

Syntax

```
#include "aclapi.h"
NSAPI_PUBLIC
void nserrFFree(NSErr_t *errp, NSEFrame_t *efp);
```

Parameters This function has the following parameters:

`errp`	Pointer to the header for a list of error frames. This argument can be NULL.
`efp`	Pointer to the error frame structure that you want to free from memory.

Example The following example frees an error frame.

```
#include "aclapi.h"

...

NSErr_t err;
NSEFrame_t *efp;

...

/* Free this frame */
nserrFFree(&err, efp);

...
```

See Also `nserrFAlloc()`, `nserrGenerate()`, `nserrDispose()`.

nserrGenerate()

The `nserrGenerate()` function generates an error frame and adds it to the specified list of error frames.

Syntax

```
#include "aclapi.h"
NSAPI_PUBLIC
NSEFrame_t *nserrGenerate(NSErr_t *errp, long retcode,
   long errorid, char *program, int errc, ...);
```

Parameters This function has the following parameters:

`errp`	Pointer to the header for a list of error frames. This argument can be NULL.
`retcode`	Error return code identifying the type of error that occurred. In the newly created error frame, the `retcode` field is set to this value. Pass `ACLERRFAIL` as the value of this argument to indicate that the error was related to ACLs.
`errorid`	Error ID which uniquely identifies errors in a module or library. In the newly created error frame, the `errorid` field is set to this value.
`program`	String value that identifies the library or module context for the `errorid` argument. In the newly created error frame, the `ef_program` field is set to this value.
`errc`	Number of subsequent arguments for the error message. In the newly created error frame, the `ef_errc` field is set to this value.
`...`	String values relevant to the current error ID. In the newly created error frame, the `ef_errv` field is set to this value.

Returns If successful, returns a pointer to the new error frame, filled with the information provided. If unsuccessful, returns NULL.

Example The following example generates a new error frame.

```
#include "aclapi.h"

#define ACLERRINVAL -12 /* invalid argument */
#define ACLERR5700 5700 /* wrong attribute */

...

NSErr_t err;
char *ACL_Program = "NSACL";
char *myAttrName = "email";
char *errorStr = "Incorrect attribute name";

...
```

```
if (strcmp(attr_name, myAttrName) != 0) {
   nserrGenerate(&err, ACLERRINVAL, ACLERR5700, ACL_Program, 2,
      errorStr, attr_name);
   return LAS_EVAL_INVALID;
}

...
```

See Also nserrFAlloc(), nserrFFree(), nserrDispose().

PListAssignValue()

The `PListAssignValue()` function sets the value and (optionally) the type of a given property. The property that is manipulated in this function is identified by property name.

To set the value and type of a property identified by property index, call the `PListSetValue()` and `PListSetType()` functions instead.

Syntax

```
#include "aclapi.h"
NSAPI_PUBLIC
int PListAssignValue(PList_t plist, const char *pname,
   const void *pvalue, PList_t ptype);
```

Parameters This function has the following parameters:

`plist`	Property list containing the property that you want to set the value and type of.
`pname`	Name of the property that you want to set the value and type of.
`pvalue`	Pointer to the value that you want assigned to the property.
`ptype`	Type that you want assigned to the property. If you specify NULL as the value of this argument, the type currently assigned to the property is left unchanged.

Returns If successful, returns the property index of the property to which you assigned the value and type. If unsuccessful, returns a value less than 0. Possible error codes returned include:

- `ERRPLUNDEF` (if the specified property list does not exist)

Example The following example assigns the value `myValue` to the property named `myName`.

```
#include "aclapi.h"

PList_t plist;
int rv;

...

rv = PListAssignValue(plist, "myName", "myValue", NULL);

...
```

See Also `PListDefProp()`, `PListSetValue()`.

PListCreate()

The `PListCreate()` function creates a new property list and reserves a set number of property list indices.

You can call this function instead of the `PListNew()` function if you want to create a property list that has a set of reserved or well-known properties at the beginning of the property list (from indices 1 through *x*). To assign values to these reserved properties, call the `PListSetValue()` function.

When you are done working with the property list, you can free it from memory by calling the `PListDestroy()` function.

Syntax

```
#include "aclapi.h"
NSAPI_PUBLIC
PList_t PListCreate(pool_handle_t *mempool, int resvprop,
   int maxprop, int flags);
```

Parameters This function has the following parameters:

`mempool`	Pointer to the memory pool from which you want the memory allocated for this property list. If `NULL`, memory for the property list is allocated from the default memory pool.
`resvprop`	Number of property indices to reserve. The property indices between 1 and this value are reserved.

`maxprop`	Maximum number of property values to be stored in the property list. If you specify a value less than or equal to 0, the size of the property list is limited only by the amount of dynamic memory available.
`flags`	Reserved for future use. (Set this to 0.)

Returns If successful, returns a handle to the new property list. If unsuccessful, returns `NULL`.

Example The following example creates a new property list with the maximum number of indexes.

```
#include "aclapi.h"

PList_t plist;

...

plist = PListCreate(NULL, ACL_ATTR_INDEX_MAX, 0, 0);

...
```

See Also `PListNew()`, `PListSetValue()`, `PListDestroy()`.

PListDefProp()

The `PListDefProp()` function creates a new property in a specified property list. This function does not assign a value to the property. To assign a value, call the `PListSetValue()` function.

If you want to create the property and assign a value in a single function call, you can call the `PListInitProp()` function.

Syntax

```
#include "aclapi.h"
NSAPI_PUBLIC
int PListDefProp(PList_t plist, int pindex, const char *pname,
   const int flags);
```

Parameters This function has the following parameters:

`plist`	Property list to which you want to add the new property.
`pindex`	Property index that you want assigned to the new property. If a property already exists at this index, the function returns an `ERRPLPEXIST` error. If you specify 0 as the value for this argument, the next available index is assigned to the property.
`pname`	Name that you want assigned to the new property. If you specify NULL as the value for this argument, no name is assigned to the property.
`flags`	Reserved for future use. (Set this to 0.)

Returns If successful, returns the property index of the new property. If unsuccessful, returns a value less than 0. Possible error codes returned include:

- `ERRPLUNDEF` (if the specified property list does not exist)
- `ERRPLNOMEM` (if not enough memory is available for this operation)
- `ERRPLPEXIST` (if `pindex` is specified as a value greater than 0 and a property already exists at that index)
- `ERRPLINVPI` (if `pindex` is specified as a value outside the bounds of the reserved range of property indices)
- `ERRPLFULL` (if the property list is already full)

Example The following example assigns the property name `myName` to the next available index.

```
#include "aclapi.h"

PList_t plist;
int rv;

...

rv = PListDefProp(plist, 0, "myName", 0);

...
```

See Also `PListSetValue()`, `PListInitProp()`.

PListDeleteProp()

The `PListDeleteProp()` function deletes a property from the specified property list.

You can identify the property to delete either by property index or by property name:

- If you specify a property index, the property identified by that index is deleted.
- If you specify 0 for the property index, the property identified by the name is deleted.

Deleting a property from a property list does not free the property value from memory.

Syntax

```
#include "aclapi.h"
NSAPI_PUBLIC
const void *PListDeleteProp(PList_t plist, int pindex,
   const char *pname);
```

Parameters This function has the following parameters:

`plist`	Property list containing the property that you want to delete.
`pindex`	Index of the property that you want to delete. If you specify 0 as the value of this argument, the value of the `pname` argument is used to find the property to delete.
`pname`	Name of the property that you want to delete. If you specify a valid property index for the `pindex` argument, that index is used to find the property to delete.

Example The following example deletes the property named `myName` from the property list.

```
#include "aclapi.h"

PList_t plist;

...

PListDeleteProp(plist, 0, "myName");

...
```

See Also `PListCreate()`, `PListNew()`.

PListDestroy()

The `PListDestroy()` function frees a property list from memory. Any dynamic memory allocated by this list is freed, but property value data is not freed from memory.

You can also free a property list from memory by freeing its associated memory pool.

Syntax

```
#include "aclapi.h"
NSAPI_PUBLIC
void PListDestroy(PList_t plist);
```

Parameters This function has the following parameters:

`plist`	Property list that you want to free from memory.

Example The following example frees a property list from memory.

```
PList_t plist;
...
PListDestroy(plist);
...
```

See Also `PListCreate()`, `PListNew()`.

PListDuplicate()

The `PListDuplicate()` function duplicates a property list. The new property list has the same property names, values, types, and indices as the original property list.

You can either associate a new memory pool with the newly created property list, or you can associate the original property list's memory pool with this new property list.

Syntax

```
#include "aclapi.h"
NSAPI_PUBLIC
PList_t PListDuplicate(PList_t plist,
   pool_handle_t *new_mempool, int flags);
```

Parameters This function has the following parameters:

`plist`	Property list that you want to duplicate.
`new_mempool`	Pointer to a new memory pool from which the memory will be allocated for this property list.
`flags`	To use the new memory pool identified by the `new_mempool` argument, specify `PLFLG_NEW_MPOOL` as the value of this argument. To use the existing memory pool associated with the original property list, specify `PLFLG_OLD_MPOOL` as the value of this argument.

Returns If successful, returns a handle to the new property list. If unsuccessful, returns `NULL`.

Example The following example duplicates a property list and its properties.

```
#include "aclapi.h"

PList_t oldplist, newplist;

...

newplist = PListDuplicate(oldplist, NULL, PLFLG_OLD_MPOOL);

...
```

See Also `PListDefProp()`, `PListSetValue()`.

PListEnumerate()

The `PListEnumerate()` function iterates through each property in a property list and calls a user-defined function of the type `PListFunc_t`.

When you call the `PListEnumerate()` function, your user-defined function is called for each property in the list. The following parameters are passed to your user-defined function in this order:

1. the name of the property
2. the value of the property
3. the data passed in to the 3rd argument of the `PListEnumerate()` function.

Syntax

```
#include "aclapi.h"
```

```
NSAPI_PUBLIC
void PListEnumerate(PList_t plist, PListFunc_t *user_func,
   void *user_data);
```

Parameters This function has the following parameters:

`plist`	Property list that you want to iterate through.
`user_func`	Pointer to the function that you want called for each property name and value
`user_data`	Pointer to data that you want passed to your function.

PListFindValue()

The `PListFindValue()` function gets the value and type of a property. The property is identified by property name.

To find the value and type of a property identified by property index, call the `PListGetValue()` function instead.

Syntax

```
#include "aclapi.h"
NSAPI_PUBLIC
int PListFindValue(PList_t plist, const char *pname,
   void **pvalue, PList_t *type);
```

Parameters This function has the following parameters:

`plist`	Property list containing the property that you want to find the value of.
`pname`	Name of the property that you want to find the value of.
`pvalue`	Location where the value of the property is returned, if the value is not NULL.
`type`	Location where the type of the property is returned, if the value is not NULL.

Returns If successful, returns the property index of the property from which you retrieved the value and type. If unsuccessful, returns a value less than 0. Possible error codes returned include:

- `ERRPLUNDEF` (if the specified property list does not exist)

Example The following example finds the value of the property named `myName`.

```
#include "aclapi.h"

PList_t plist;
char *myValue;
int rv;

...

rv = PListFindValue(plist, "myName", (void **)&myValue, NULL);

...
```

See Also PListGetValue().

PListFunc_t

PListFunc_t is the type definition for a function. This type describes a kind of callback function that is called to iterate through each property in a property list.

Syntax

```
#include "aclapi.h"
NSAPI_PUBLIC
#ifdef __cplusplus
typedef void (PListFunc_t)(char*, const void*, void*);
#else
typedef void (PListFunc_t)();
#endif
```

Parameters See PListEnumerate() for a description.

See Also PListEnumerate().

PListGetPool()

The PListGetPool() function gets the memory pool used by the specified property list.

Syntax

```
#include "aclapi.h"
NSAPI_PUBLIC
pool_handle_t *PListGetPool(PList_t plist);
```

Parameters This function has the following parameters:

plist	Property list that allocates memory from the memory pool that you want to get.

Returns This function returns the memory pool address, which can be NULL.

Example The following example gets the memory pool used by a property list.

```
#include "aclapi.h"

PList_t plist;
pool_handle_t *phandle;

...

phandle = PListGetPool(plist);

...
```

See Also PListCreate(), PListDuplicate(), PListNew().

PListGetValue()

The `PListGetValue()` function gets the value and type of a property. The property is identified by property index.

To find the value and type of a property identified by property name, call the PListFindValue() function instead.

Syntax

```
#include "aclapi.h"
NSAPI_PUBLIC
int PListGetValue(PList_t plist, int pindex, void **pvalue,
PList_t *type);
```

Parameters This function has the following parameters:

`plist`	Property list containing the property that you want to find the value of.
`pindex`	Index of the property that you want to find the value of.
`pvalue`	Location where the value of the property is returned, if the value is not NULL.
`type`	Location where the type of the property is returned, if the value is not NULL.

Returns If successful, returns the property index of the property from which you retrieved the value and type. If unsuccessful, returns a value less than 0. Possible error codes returned include:

- `ERRPLUNDEF` (if the specified property list does not exist)
- `ERRPLINVPI` (if `pindex` is specified as a value outside the bounds of the reserved range of property indices)

Example The following example gets the pointer to the NSAPI request structure from the resource property list.

```
#include "aclapi.h"

PList_t resource;
Request *rq;
int rv;

...

rv = PListGetValue(resource, ACL_ATTR_REQUEST_INDEX,
   (void **)&rq, NULL);

...
```

See Also `PListFindValue()`, `PListSetType()`, `PListSetValue()`.

PListInitProp()

The `PListInitProp()` function creates a new property in a specified property list and assigns a value for that property. Calling this function is equivalent to calling the `PListDefProp()` function followed by the `PListSetValue()` function.

Syntax

```
#include "aclapi.h"
NSAPI_PUBLIC
int PListInitProp(PList_t plist, int pindex, const char *pname,
   const void *pvalue, PList_t ptype);
```

Parameters This function has the following parameters:

`plist`	Property list to which you want to add the new property.
`pindex`	Property index that you want assigned to the new property. If a property already exists at this index, the function returns an `ERRPLPEXIST` error. If you specify 0 as the value for this argument, the next available index is assigned to the property.
`pname`	Name that you want assigned to the new property. If you specify NULL as the value for this argument, no name is assigned to the property.

`pvalue`	Pointer to the value that you want assigned to the new property.
`ptype`	Type that you want assigned to the new property. If you specify NULL as the value of this argument, no type is assigned to the property.

Returns If successful, returns the property index of the new property. If unsuccessful, returns a value less than 0. Possible error codes returned include:

- `ERRPLUNDEF` (if the specified property list does not exist)
- `ERRPLNOMEM` (if not enough memory is available for this operation)
- `ERRPLPEXIST` (if `pindex` is specified as a value greater than 0, and a property already exists at that index)
- `ERRPLINVPI` (if `pindex` is specified as a value outside the bounds of the reserved range of property indices)
- `ERRPLFULL` (if the property list is already full)

Example The following example sets the referer property in the subject property list. (The referer property is the URL that referred the client to the current URL.)

```
#include "aclapi.h"

PList_t subject;
char *referer="http://home.netscape.com/";
int rv;

...

/* Add the referer to the subject property list */
rv = PListInitProp(subject, 0, LASREF_PROPNAME, STRDUP(referer), NULL);

...
```

See Also `PListDefProp()`, `PListSetValue()`.

PListNameProp()

The `PListNameProp()` function assigns a name to a defined property. If the property already has a name, the existing name is replaced with the new name.

Syntax `#include "aclapi.h"`

```
NSAPI_PUBLIC
int PListNameProp(PList_t plist, int pindex, const char *pname);
```

Parameters This function has the following parameters:

`plist`	Property list containing the property that you want to assign the name to.
`pindex`	Property index of the property that you want to assign the name to.
`pname`	Name that you want assigned to the property.

Returns If successful, returns the property index of the property. If unsuccessful, returns a value less than 0. Possible error codes returned include:

- `ERRPLUNDEF` (if the specified property list does not exist)
- `ERRPLINVPI` (if `pindex` is specified as a value outside the bounds of the reserved range of property indices)

Example The following example assigns the name `myName` to the first property.

```
#include "aclapi.h"

PList_t plist;
int rv;

...

rv = PListNameProp(plist, 1, "myName");

...
```

See Also `PListDefProp()`, `PListInitProp()`.

PListNew()

The `PListNew()` function creates a new, empty property list in memory.

After you create a new property list, you can create new properties and values for the list by calling the `PListInitProp()` function or the `PListDefProp()` and `PListSetValue()` functions.

When you are done working with the property list, you can free it from memory by calling the `PListDestroy()` function.

Syntax

```
#include "aclapi.h"
NSAPI_PUBLIC
PList_t PListNew(pool_handle_t *mempool);
```

Parameters This function has the following parameters:

`mempool`	Pointer to the memory pool from which you want the memory allocated for this property list. If `NULL`, memory for the property list is allocated from the default memory pool.

Returns If successful, returns a handle to the new property list. If unsuccessful, returns `NULL`.

Example The following example creates a new property list.

```
#include "aclapi.h"

PList_t plist;

...

plist = PListNew(NULL);

...
```

See Also `PListCreate()`, `PListInitProp()`, `PListDefProp()`, `PListSetValue()`, `PListDestroy()`.

PListSetType()

The `PListSetType()` function sets the type of a property. The property is identified by property index.

To set the type of a property identified by property name, call the `PListAssignValue()` function instead.

To set both the type and value of a property identified by property index, call the `PListSetValue()` function instead.

Syntax

```
#include "aclapi.h"
NSAPI_PUBLIC
int PListSetType(PList_t plist, int pindex, PList_t type);
```

Parameters This function has the following parameters:

`plist`	Property list containing the property that you want to set the type of.
`pindex`	Index of the property that you want to set the type of.
`type`	Type that you want assigned to the property. If NULL, the type of the property is set to undefined.

Returns If successful, returns the property index of the property for which you set the type. If unsuccessful, returns a value less than 0. Possible error codes returned include:

- `ERRPLUNDEF` (if the specified property list does not exist)
- `ERRPLINVPI` (if `pindex` is specified as a value outside the bounds of the reserved range of property indices)

See Also `PListAssignValue()`, `PListSetValue()`, `PListGetValue()`.

PListSetValue()

The `PListSetValue()` function sets the value and type of a property. The property is identified by property index.

To set the value and type of a property identified by property name, call the `PListAssignValue()` function instead.

If you only want to set the type of the property, call the `PListSetType()` function instead.

Syntax

```
#include "aclapi.h"
NSAPI_PUBLIC
int PListSetValue(PList_t plist, int pindex, const void *pvalue,
   PList_t type);
```

Parameters This function has the following parameters:

`plist`	Property list containing the property that you want to set the value and type of.
`pindex`	Index of the property that you want to set the value and type of.

`pvalue`	Pointer to the value that you want assigned to the property.
`type`	Type that you want assigned to the property. If you specify NULL as the value of this argument, the type currently assigned to the property is left unchanged.

Returns If successful, returns the property index of the property for which you set the value and type. If unsuccessful, returns a value less than 0. Possible error codes returned include:

- `ERRPLUNDEF` (if the specified property list does not exist)
- `ERRPLINVPI` (if `pindex` is specified as a value outside the bounds of the reserved range of property indices)

See Also `PListGetValue()`, `PListAssignValue()`, `PListSetType()`.

Appendix

A

ACL Files

This appendix describes access control list (ACL) files and their syntax. ACL files are text files that contain lists that define who can access resources stored on your web server. By default, the web server uses one ACL file that contains all of the lists for access to your server. However, you can create multiple ACL files and reference them in the `obj.conf` file.

You need to know the syntax and function of ACL files if you plan on customizing access control using the Access Control API. For example, you might use the Access Control API to interface with another database, such as an Oracle or Informix database. For more information on the API, see the DevEdge site at:

```
http://developer.netscape.com/library/documentation/index.html
```

ACL File Syntax

All ACL files must follow a specific format and syntax. An ACL file is a text file containing one or more ACLs. All ACL files must begin with the version number they use. There can be only one version statement and it can appear after any comment lines. For example:

```
version 3.0;
```

You can include comments in the file by beginning a comment line with the # sign.

Each ACL in the file begins with a statement that defines its type and identifies the resource to which the ACL applies. ACLs can be one of three types:

- Path ACLs specify an absolute path to the resource they affect.
- URI (Uniform Resource Identifier) ACLs specify a directory or file relative to the server's document root.
- Named ACLs specify a name that is referenced in resources in the `obj.conf` file. The server comes with a "default" named resource that allows read access to anyone and write access to users in the local database or LDAP directory. Even though you can create a named ACL from the Server Manager forms, you must manually reference the named ACLs with resources in the `obj.conf` file.

The ACL begins with the keyword `acl` and then includes the type and resource information in double-quotation marks followed by a semicolon. The type and resource information for each ACL must be a unique—even among different ACL files. The following lines show different types of ACLs:

```
acl "path=C:\Netscape\SuiteSpot\docs\mydocs\";
acl "uri=/mydocs/";
acl "*.html";
acl "default";
```

After you specify the type and resource for an ACL, you can have one or more statements that define the method used with the ACL (authentication statements) and the people and computers who are allowed or denied access (authorization statements). The following sections describe the syntax for these statements.

Authentication Statements

ACLs can optionally specify the authentication method the server must use when processing the ACL. There are two general methods:

- Basic requires users to enter a username and password before accessing a resource.
- SSL requires the user to have a client certificate. For this method to work, the web server must have encryption turned on.

By default, the server uses the Basic method for any ACL that doesn't specify a method. You can change the default setting by editing the following line in the `magnus.conf` file:

```
Init fn=acl-set-default-method method=SSL
```

Each authentication statement begins with the keyword `authenticate` and must specify what list (users, groups, or both) the server should use when authenticating users. The following authentication statement, which would appear after the ACL type line, specifies basic authentication with users matched to individual users in the database or directory:

```
authenticate (user) {
    method = basic;
};
```

The following example uses SSL as the authentication method for users and groups:

```
authenticate (user, group) {
    method = ssl;
};
```

Warning In a given ACL, any allow or deny statements (described in the following section) must match the lists you specify in the authentication statement. That is, if the statement says `authenticate (user)`, the allow or deny statement must also specify `user`. The following example allows any user whose username begins with the letters `sales`:

```
authenticate (user);
allow (all)
    user = sales*;
```

In the last line, if `user` was changed to `group`, then the ACL would fail because there are no groups in the user lists.

Authorization Statements

An ACL can include one or more authorization statements. Authorization statements specify who is allowed or denied access to a server resource. Use the following syntax when writing authorization statements:

```
allow|deny [absolute] [static] [content] (right[,right...])
    attribute qualifier expression;
```

Start each statement with either the `allow` or `deny` keyword.

It is usually a good idea to deny access to anyone in the first rule and then specifically allow access for users, groups, or computers in subsequent rules. This is because of the hierarchy of rules. If you allow anyone access to a directory called `/my_stuff`, and then you have a subdirectory `/my_stuff/personal` that allows access to a few users, the access control on the subdirectory won't work because anyone allowed access to the `/my_stuff` directory will also be allowed access to the `/my_stuff/personal` directory. To prevent this, create a rule for the subdirectory that first denies access to anyone and then allows it for the few users who need access.

However, in some cases if you set the default ACL to deny access to anyone, then your other ACL rules don't need a "deny all" rule.

The following authorization statement denies access to anyone:

```
deny (all)
    user = "anyone";
```

Hierarchy of Authorization Statements

ACLs have a hierarchy that depends on the resource. For example, if the server receives a request for the document (URI) `/my_stuff/web/presentation.html`, the server first looks for an ACL that matches the file type or any other wildcard pattern that matches the request, then it looks for an ACL on the directory, and finally it looks for an ACL on the URI. If there is more than one ACL that matches, the server uses the last statement that matches. However, if you use an absolute statement, then the server stops looking for other matches and uses the ACL containing the absolute statement. If you have two absolute statements for the same resource, the server uses the first one in the file and stops looking for other resources that match.

For example, using the ACL hierarchy with the request for the document `/my_stuff/web/presentation.html`, you could have an absolute ACL that restricts access to the file type `*.html`. Then the server would use that ACL instead of looking for one that matches the path or URI.

```
version 3.0;

acl "default";
authenticate (user,group) {
    prompt="Enterprise Server";
};
allow (read,execute,list,info)
    user = "anyone";
```

```
allow (write,delete)
    user = "all";

acl "*.html";
deny absolute (all)
    user = "anyone";

acl "uri=/my_stuff/web/presentation.html";
deny (all)
    user = "anyone";
allow (all)
    user = "anyone";
```

You can also flag an authorization statement as `static` or `content`. A static statement applies directly to a container and not to any of its contents. A content statement applies to the contents of a container but not to the container itself. For more details, see Table 9.14, "Bit flags that you can apply to access control entries," on page 148.

Attribute Qualifier Expressions

Attribute qualifier expressions define who is allowed or denied access based on their username, group name, host name, or IP address. The following lines are examples of allowing access to different people or computers:

- user = "all"
- user = "anyone"
- user = "smith*"
- group = "sales"
- dns = "*.netscape.com"
- dns = "*.netscape.com" or "*.mozilla.com"
- ip = "198.*"

Note Specifying `user = "all"` is different than specifying `user = "anyone"`; they have different semantics. In `user = "all"`, the `"all"` means "all authenticated users," which directs the server to require authentication first. Given this authorization statement:

```
deny (all) (user = all);
```

the server will force all incoming clients to authenticate. In `user = "anyone"`, the `"anyone"` means "any unauthenticated user," so the server will not require authentication for every client.

You can also restrict access to your server by time of day (based on the local time on the server) by using the `timeofday` attribute qualifier. For example, you can use the `timeofday` attribute qualifier to restrict access to certain users during specific hours.

Note Use 24-hour time to specify times (for example, use 0400 to specify 4 a.m. or 2230 for 10:30 p.m.).

The following statement restricts access to a group of users called *guests* between 8 a.m. and 4:59 pm.

```
allow (read)
    (group="guests") and
    (timeofday<800 or timeofday>=1700);
```

You can also restrict access by day of the week. Use the following three-letter abbreviations to specify days of the week: Sun, Mon, Tue, Wed Thu, Fri, and Sat.

The following statement allows access for users in the *premium* group any day and any time. Users in the *discount* group get access all day on weekends and on weekdays anytime except 8am-4:59pm.

```
allow (read)
    (group="discount" and dayofweek="Sat,Sun") or
    (group="discount" and (dayofweek="mon,tue,wed,thu,fri" and
        (timeofday<0800 or timeofday>=1700))) or
    (group="premium");
```

Operators for Expressions

You can use various operators in attribute qualifier expressions. You can use parentheses to delineate the order of precedence of the operators. With `user`, `group`, `dns`, and `ip` qualifiers, you can use the following operators:

- `and`
- `or`
- `not`
- `= (equals)`

- `!= (not equal to)`

With `timeofday` and `dayofweek` qualifiers, you can use the following additional operators:

- `>` greater than
- `<` less than
- `>=` greater than or equal to
- `<=` less than or equal to

General Syntax Items

Input strings can contain the following characters:

- Letters a through z
- Numbers 0 through 9
- Period and underscore

If you use any other characters, you need to use double-quotation marks around the characters.

A single statement can be placed on its own line and be terminated with a semicolon. Multiple statements are placed within braces. A list of items must be separated by commas and enclosed in double-quotation marks (see the dayofweek example).

The Default ACL File

After installing the server, the server uses the default settings in the file *`server_root`*`/httpacl/generated.https-`*`server_id`*`.acl`. There is also a file called `genwork.https-`*`server_id`*`.acl` that is a working copy the server uses until you save and apply your changes when working with the user interface. When editing the ACL file, you might want to work in the genwork file and then use the Server Manager to save and apply the changes.

The following text is from the default file:

```
# File automatically written
#
# You may edit this file by hand
#

version 3.0;

acl "agents";
authenticate (user,group) {
    prompt = "Enterprise Server";
};
deny (all)
    user = "anyone";
allow absolute (all)
    user = "all";

acl "default";
allow (read,execute,list,info)
    user = "anyone";
allow (write,delete)
    user = "all";
```

The default ACL file is referenced in `magnus.conf` as follows:

```
ACLFile absolute_path/generated.https-server_id.acl
```

You can reference multiple ACL files in `magnus.conf` and then use their ACLs for resources in `obj.conf`. However, the server uses only the first ACL file with the web publisher and when evaluating access control for objects that don't have specific ACLs listed in `obj.conf`. If you are using the Server Manager forms to do some access control, the first ACL file in `magnus.conf` should point to the file `generated.https-server_id.acl`. See Referencing ACL Files in obj.conf for more information.

Referencing ACL Files in obj.conf

If you have named ACLs or separate ACL files, you can reference them in the `obj.conf` file. You do this in the `PathCheck` directive using the `check-acl` function. The line has the following syntax:

```
PathCheck fn="check-acl" acl="aclname"
```

The `aclname` is a unique name of an ACL as it appears in any ACL file.

For example, you might add the following lines to your `obj.conf` file if you want to restrict access to a directory using the acl named `testacl`:

```
<Object ppath="/usr/ns-home/docs/test/*">
PathCheck fn="check-acl" acl="testacl"
</Object>
```

In the previous example, the first line is the object that states which server resource you want to restrict access to. The second line is the `PathCheck` directive that uses the `check-acl` function to bind the name ACL (`testacl`) to the object in which the directive appears. The `testacl` ACL can appear in any ACL file referenced in `magnus.conf`.

Index

A

B

C

D

E

F

G

H

I

L

M

N

O

P

R

S

T

U

V