IBM® SecureWay® Policy Director

Authorization API: Java Reference

Version 3.0.1

IBM® SecureWay® Policy Director

Authorization API: Java Reference

Version 3.0.1

Note

Before using this information and the product it supports, read the general information under "Appendix. Notices" on page 83.

First Edition (January 2000)

This edition applies to Version 3, release 0, modification 1 of IBM SecureWay Policy Director product and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

About this book

This book contains programming guide and reference information about the Java implementation of the IBM SecureWay Policy Director Authorization application programming interface (API).

Who should read this book

Developers who are designing and developing applications for IBM SecureWay Policy Director should read this book.

Developers should have some knowledge of IBM Distributed Computing Environment (DCE) and the IBM SecureWay Directory's lightweight directory access protocol (LDAP). DCE and LDAP are co-requisite products of Policy Director. Developers should have basic working knowledge about writing and configuring DCE and LDAP servers.

Developers should also have knowledge of Java programming.

How this book is organized

This book contains the following chapters:

- "Chapter 1. IBM SecureWay" on page 1 introduces you to the IBM SecureWay FirstSecure and IBM SecureWay Policy Director products.
- "Chapter 2. Introducing the Authorization API" on page 3 introduces the Authorization API and describes the Policy Director Java implementation.
- "Chapter 3. Using the Authorization API" on page 15 guides the application designer or developer on the use of the Policy Director Authorization API.
- "Chapter 4. Deploying applications with the Authorization API" on page 33 describes the requirements for deploying applications with the Authorization API.
- "Chapter 5. Authorization API: Java Reference" on page 35 provides reference information about the Policy Director Authorization Java API.

What is new in this release

On the IBM SecureWay Policy Director Version 3.0.1 CD, you will find:

- Code updates and fixes to the Version 3.0 product released in October 1999.
- A README file, Version 3.0.1, in Hypertext Markup Language (HTML) format (PD301_csd_readme.html).
- An IBM SecureWay Policy Director Administration Guide: Additions and Corrections, Version 3.0.1
- An IBM SecureWay Policy Director Programming Guide, Version 3.0.1
- All IBM SecureWay Policy Director documentation for Version 3.0 that was released in October 1999.

  See the *Policy Director Up and Running* book, which provides information about what is new for Version 3.0 of IBM SecureWay Policy Director.

At the IBM SecureWay Policy Director Web site, you will find:

- The IBM SecureWay Policy Director Migration Guide and related migration files for AIX, Solaris, or Windows NT.
- The IBM SecureWay Policy Director Authorization API: Java Reference software and documentation.
- The IBM SecureWay Policy Director Quick Installation Guide for Windows NT.

See “Web information” for related Web addresses.

Year 2000 readiness

This product is Year 2000 ready. When used in accordance with its associated documentation, it is capable of correctly processing, providing, and/or receiving date data within and between the twentieth and twenty-first centuries, provided that all products (for example, hardware, software, and firmware) used with the products properly exchange accurate date data with it.

Service and support

Contact IBM for service and support for all the products included in the IBM SecureWay FirstSecure offering. Some of these products might refer to non-IBM support. If you obtain these products as part of the FirstSecure offering, contact IBM for service and support.

Conventions

This book uses the following typographical conventions:

Convention	Meaning
bold	User interface elements such as check boxes, buttons, and items inside list boxes.
`monospace`	Syntax, sample code, and any text that the user must type.
Italic	Emphasis and first use of special terms that are relevant to Policy Director.
>	Shows a series of selections from a menu. For example, click **File > Run** means click **File**, and then click **Run**.

Web information

Information about last-minute updates to Policy Director is available at the following Web address:

`http://www.ibm.com/software/security/policy/library`

You can download the Policy Director Authorization Java API source files and other Policy Director source files from this Web address:

`http: //www.ibm.com/software/security/policy/downloads`

Information about updates to other IBM SecureWay FirstSecure products is available by starting at the following Web address:

`http://www.ibm.com/software/security/firstsecure/library`

Chapter 1. IBM SecureWay

IBM SecureWay Policy Director (Policy Director) is available either as a component of IBM SecureWay FirstSecure or as a standalone product.

What is IBM SecureWay FirstSecure?

IBM SecureWay FirstSecure (FirstSecure) is part of the IBM integrated security solution. FirstSecure is a comprehensive set of integrated products that help your company:

- Establish a secure e-business environment.
- Reduce the total cost of security ownership by simplifying security planning.
- Implement security policy.
- Create an effective e-business environment.

The IBM SecureWay products include:

Policy Director
IBM SecureWay Policy Director (Policy Director) provides authentication, authorization, data security, and Web resource management.

Boundary Server
IBM SecureWay Boundary Server (Boundary Server) provides:

- The critical firewall functions of filtering, proxy, and circuit level gateway
- A virtual private network (VPN) connection to the IBM Firewall
- The components for Internet security
- A mobile code security solution

A configuration graphical user interface (GUI) ties together the Policy Director's proxy user function with the Boundary Server's Firewall product.

Intrusion Immunity
Intrusion Immunity provides intrusion detection and antivirus protection.

Trust Authority
IBM SecureWay Trust Authority (Trust Authority) supports public key infrastructure (PKI) standards for cryptography and interoperability. Trust Authority provides support for issuance, renewal, and revocation of digital certificates. These certificates provide a means to authenticate users and to ensure trusted communications.

Toolbox
The IBM SecureWay Toolbox (Toolbox) is a set of application programming interfaces (API) with which application programmers can incorporate security into their software. You can obtain the Toolbox as part of FirstSecure. Both Policy Director and the Toolbox include the Policy Director API library and documentation. The Toolbox README file contains installation instructions for the Policy Director ADK.

Because each IBM SecureWay FirstSecure product can be installed independently, you can plan a controlled move toward a secure environment. This capability reduces the complexity and cost of securing your environment and speeds deployment of Web applications and resources.

See the FirstSecure *Planning and Integration* documentation for more information about the FirstSecure components and for a list of all the IBM SecureWay products' documentation.

What is IBM SecureWay Policy Director?

Policy Director is a standalone authorization and security management solution. Policy Director provides end-to-end security of resources over geographically dispersed intranets and *extranet*s. An *extranet* is a virtual private network (VPN) that uses access control and security features to restrict the use of one or more intranets attached to the Internet to selected subscribers.

Policy Director provides authentication, authorization, data security, and resource-management services. You can use Policy Director in conjunction with standard Internet-based applications to build secure and well-managed intranets and extranets.

Policy Director runs on the Windows NT, AIX, and Solaris operating systems.

Chapter 2. Introducing the Authorization API

This chapter includes:

Accessing the Policy Director authorization service

Using the Policy Director Authorization Application Programming Interface (API), you can code Policy Director applications and third-party applications to query the Policy Director Authorization Service for authorization decisions.

The Policy Director Authorization API is the interface between the server-based resource manager and the authorization service and provides a standard model for coding authorization requests and decisions. The Authorization API lets you make standardized calls to the centrally managed authorization service from any legacy or newly developed application.

The Authorization API supports two implementation modes:

- **Remote cache mode**

  In remote cache mode, you use the Authorization API to call the Policy Director Authorization Server, which performs authorization decisions on behalf of the application. The Authorization Server maintains its own cache of the replica authorization policy database.

- **Local cache mode**

  In local cache mode, you use the Authorization API to download a local replica of the authorization policy database. In this mode, the application can perform all authorization decisions locally.

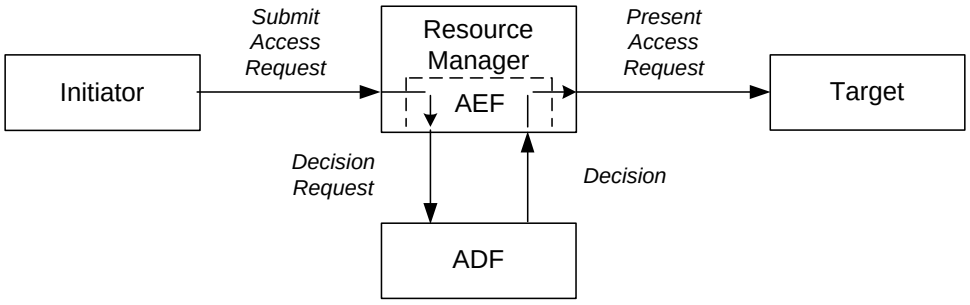
The Authorization API shields you from the complexities of the authorization service mechanism. Issues of management, storage, caching, replication, credentials format, and authentication methods are all hidden behind the Authorization API.

The Authorization API works independently from the underlying security infrastructure, the credential format, and the evaluating mechanism. The Authorization API makes it possible to request an authorization check and get a simple “yes” or “no” recommendation in return.

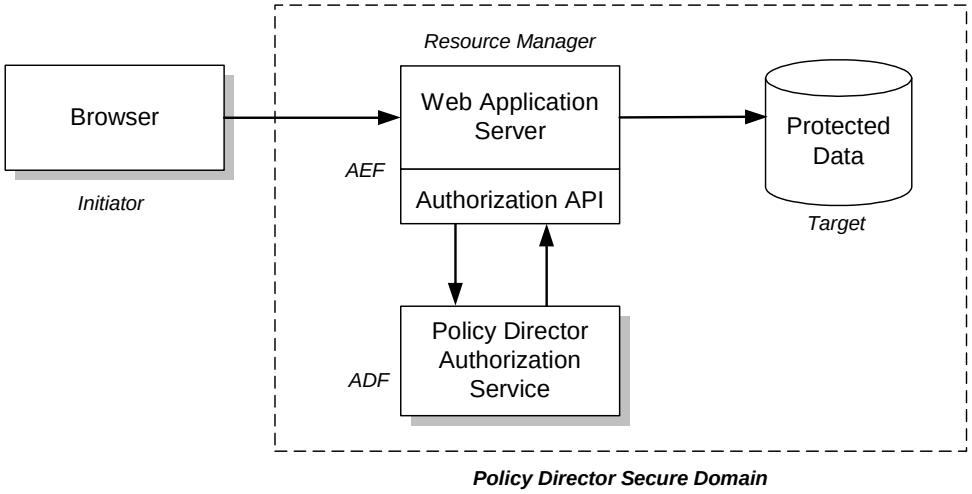
The Authorization API is a component of the Policy Director Application Development Kit (ADK).

The Open Group Authorization API standard

The Policy Director Authorization API implements The Open Group Authorization API (Generic Application Interface for Authorization Frameworks) standard. This interface is based on the International Organization for Standardization (ISO) 10181-3 model for authorization. In this model, an initiator requests access to a target resource. The initiator submits the request to a resource manager, which incorporates an access enforcement function (AEF). The AEF submits the request, along with information about the initiator, to an access decision function (ADF). The ADF returns a decision to the AEF, and the AEF enforces the decision.



Policy Director implements the ADF component of this model and provides the Authorization API as an interface to this function.



In the figure above, a browser (initiator) requests access to a file or other resource on a protected system (target). The browser submits the request to a Web application server (the resource manager incorporating the access enforcement function). The Web application server uses the Authorization API to submit the request to the Policy Director Authorization Service (the access decision function).

The Policy Director Authorization Service returns an access decision, through the Authorization API, to the Web application server. The Web application server processes the request as appropriate.

To implement this model, developers of AEF applications add Authorization API function calls to their application code.

Note: Developers should refer to the Open Group Authorization API document for additional information on the standard authorization model.

Policy Director Authorization API version history

This programming guide describes Policy Director Authorization Java API, Version 3.0.1. This is the first release of a Java implementation of the Open Group Authorization API standard.

Policy Director Version 3.0.1 conforms to the Open Group Authorization API, Version 1.1, published in January 2000. The Open Group Authorization API standard specifies a programming interface written in the C language. Policy Director provides both a C and a Java implementation of Version 1.1.

The C implementation of the Policy Director Authorization API is described in the *Policy Director Programming Guide and Reference, Version 3.0.1*. This document also lists the changes made to the C API between the Open Group Authorization API standard Version 1.0 (September 1999) and Version 1.1.

The previous release of Policy Director (Version 3) provided an Authorization C API that conformed to Version 1.0 of the Open Group Authorization API standard. Developers who programmed to the Version 1.0 C API might want to review the changes made for Version 1.1 before programming to the Authorization Java API.

Prior to the adoption of Version 1.0 of the Open Group Authorization API standard, IBM provided an Authorization API which was released as part of Policy Director, Version 2.1. The differences between the Version 1.0 of the Open Group Authorization and the earlier Policy Director Version 2.1 Authorization API are described in the *Policy Director Programming Guide and Reference, Version 3.0.1*.

Background and references for using Policy Director authorization

The first step in adding authorization to an application is to define the security policy requirements for your application. Defining a security policy means that you must determine the business requirements that apply to the application's users, operations, and data. These requirements include:

- Objects to be secured
- Operations permitted on each object
- Users that are permitted to perform the operations

After your security requirements have been defined, you can use the Authorization API to integrate your security policy with the Policy Director security model.

Complete the following steps in order to deploy an application into a Policy Director secure domain:

1. Configure the Policy Director secure domain to recognize and support the objects, actions, and users that are relevant to your application.
    - For an introduction to the Policy Director authorization model, see "Chapter 3, Understanding authorization" in the *Policy Director Administration Guide.*
    - For complete information on access control, see "Chapter 7, Understanding Access Control" in the *Policy Director Administration Guide.*

2. Use the Authorization API within your application to obtain the needed authorization decisions.
    - For an introduction to the Authorization API, including information on remote cache mode and local cache mode, see “Chapter 3, Understanding authorization” in the *Policy Director Administration Guide.*
3. Develop your application logic to enforce the security policy.

Installing the Java Authorization API

The Policy Director Authorization Java API is included as an optional installation package for Policy Director. The installation package is available for download on the following IBM Web site:

`http://www.ibm.com/software/security/policy/downloads`

The Authorization Java API files can be installed in any directory.

For installation instructions and a list of the files contained in the Authorization Java API, see the README file that accompanies the installation packages on the IBM Web site.

Building applications with the Authorization API

The following sections provide information on building an application with the Authorization API:

- "Installing required software" on page 6
- "Setting environment variables" on page 7

For information on the runtime requirements for applications that use the Authorization API, see "Deploying applications with the Authorization API" on page 33.

For information on using the AznDemo demonstration program, see "Running the example program AznDemo" on page 33.

Installing required software

Java Development Kit

Use the Java Development Kit 1.1.7 or later to add Authorization Java APIs to an application.

Policy Director

To develop applications that use the Policy Director Authorization API, you must install and configure a Policy Director secure domain.

If you do not have a Policy Director secure domain installed, install one before beginning application development. The minimum installation consists of a single system with the following Policy Director components installed:

- Policy Director Base (IVBase)
- Policy Director Management server (IVMgr)
- Policy Director Authorization server (IVAcld)

- Policy Director Application Development Kit (IVAuthADK)
- Policy Director Management Console (IVConsole)

If the Policy Director secure domain uses an LDAP user registry, the application development system must have an LDAP client installed.

For Policy Director installation instructions refer to the *Policy Director Up and Running* guide.

If you already have a Policy Director secure domain installed, and want to add a development system to the domain, the minimum Policy Director installation consists of the following components:

- Policy Director Base (IVBase)
- Policy Director Authorization server (IVAcld)
- Policy Director Application Development Kit (IVAuthADK)

Note: The development environment must include a DCE runtime. The DCE runtime is installed as a prerequisite to the Policy Director installations described above.

Setting environment variables

To develop applications with the Authorization Java API, set the necessary environment variables. Complete the following steps:

1. Add azn.jar to the environment variable CLASSPATH. Be sure to add the full pathname including the filename azn.jar.

   The file azn.jar containing the executable Java class for the AznDemo program.

2. Add the name of the directory containing the Authorization API Java Native Interface to the appropriate environment variable, as follows:
   - On Windows NT only, add to PATH the name of the directory containing the file aznjni.dll.
   - On AIX systems only, add to LIBPATH the name of the directory containing the file libaznjni.a.
   - On Solaris systems only, add to LD_LIBRARY_PATH the name of the directory containing the file libaznjni.so.

Introducing the Java Authorization API classes and methods

The Policy Director Java Authorization APIs are implemented as JNI native methods which invoke the corresponding C Authorization APIs. There is a one-to-one mapping between the Java Authorization APIs (the methods in the Azn class) and the C Authorization APIs. The C Authorization APIs are fully documented in the *Policy Director Programming Guide and Reference*.

The Java Authorization APIs are designed to be as close as possible to the corresponding C APIs. The function names for the C APIs all begin with azn_. The corresponding method names for the Java APIs begin with the class name Azn. For example, the C API azn_initialize corresponds to the static method Azn.initialize in the Azn class.

The parameters to the Java methods are as close as possible to the C API function parameters. An example of a small difference is where the C APIs specify pointers to output parameters, such as a pointer to an integer. In this case, an AznInteger object is passed as input to the Java method so that an integer value can returned as an output parameter. The AznAttrList, AznBuffer, AznCreds, AznString and AznStrings objects are used in a similar manner to obtain output parameters.

The following Java classes are defined:

Class com.ibm.pd.Authzn.Azn

The Azn class implements static native methods used to invoke the Policy Director Authorization APIs, which are C based APIs. There is a one-to-one mapping between the Java methods implemented by this class and the C based Authorization APIs.

Note: The C APIs are fully documented in the *Policy Director Programming Guide and Reference*.

The C based APIs all begin with azn_, while the methods in this class are named by removing the “azn_ “ portion of the C API function name and retaining the remainder of the name. For example, the C API azn_initialize function corresponds to the “initialize” method in this class. Since initialize is a static method, it is invoked using the class name Azn.initialize.

The parameters to the methods in this class correspond as closely to the parameters for the C APIs.

The following tables list the Authorization API methods and provide a reference to the section in this document that describes each method’s task.

Attribute lists

Method	Task
"Azn.attrlist_add_entry" on page 46 "Azn.attrlist_add_entry_buffer" on page 46 "Azn.attrlist_create" on page 47 "Azn.attrlist_delete" on page 47 "Azn.attrlist_get_entry_buffer_value" on page 48 "Azn.attrlist_get_entry_string_value" on page 49 "Azn.attrlist_get_names" on page 49 "Azn.attrlist_name_get_num" on page 50	"Class com.ibm.pd.Authzn.AznAttrList" on page 11

Credentials

Method	Task
"Azn.creds_combine" on page 50	"Creating a chain of credentials" on page 31
"Azn.creds_create" on page 51	"Obtaining authorization credentials for the user" on page 26
"Azn.creds_delete" on page 51	"Releasing allocated memory" on page 30
"Azn.creds_for_subject" on page 52	"Obtaining a credential from a chain of credentials" on page 31
"Azn.creds_get_attrlist_for_subject" on page 53	"Obtaining an attribute list from a credential" on page 32
"Azn.creds_get_pac" on page 53	"Converting credentials to a transportable format" on page 30
"Azn.creds_modify" on page 54	"Modifying the contents of a credential" on page 32
"Azn.creds_num_of_subjects" on page 55	"Determining the number of credentials in a credentials chain" on page 31
"Azn.id_get_creds" on page 59	"Obtaining authorization credentials for the user" on page 26
"Azn.pac_get_creds" on page 61	"Converting credentials to the native format" on page 31

Authorization decisions

Method	Task
"Azn.decision_access_allowed" on page 56	"Obtaining an authorization decision" on page 28
"Azn.decision_access_allowed_ext" on page 57	

Initialization, shutdown, and error handling

Method	Task
"Azn.error_major" on page 58	"Status codes and error handling" on page 13
"Azn.error_minor" on page 58	
"Azn.error_minor_get_string" on page 59	
"Azn.initialize" on page 60	"Initializing the authorization service" on page 16
"Azn.shutdown" on page 62	"Cleaning up and shutting down" on page 30

API extensions

Method	Task
"Azn.util_client_authenticate" on page 62	"Logging in using a password" on page 22
"Azn.util_password_authenticate" on page 63	"Obtaining an identity for a user" on page 23
"Azn.util_server_authenticate" on page 64	"Logging in using a DCE keytab file" on page 22

Class com.ibm.pd.Authzn.AznString

The AznString class implements an object used to return a string value.

An object of this class simply contains the string value which is an output parameter for the methods that return a string value.

Use the AznString objects to pass character string data between your application and the Authorization API. For example, to construct a string:

```
AznString testuser = new AznString();
testuser.value = “user_name”;
```

Class com.ibm.pd.Authzn.AznStrings

The AznStrings class implements an object used to return an array of string values.

An object of this class simply contains the string array which is an output parameter for the methods that return an array of string values.

Class com.ibm.pd.Authzn.AznInteger

The AznInteger class implements an object used to return an integer value.

An object of this class simply contains the integer value which is an output parameter for the methods that return an integer value.

Class com.ibm.pd.Authzn.AznBuffer

The AznBuffer class implements a binary buffer value. The buffer value is represented in the Authorization C APIs by the data type azn_buffer_t.

An object of this class contains a single data member which is a byte array. The byte array is used as either an input or output parameter for the Azn methods that require a buffer value.

AznBuffer objects are used as input parameters to the Azn.pac_get_creds and Azn.attrlist_add_entry_buffer methods.

An.Buffer objects are used output parameters to the Azn.attrlist_get_entry_buffer_value and Azn.creds_get_pac methods.

Class com.ibm.pd.Authzn.AznAttrList

The AznAttrList class implements an attribute list. Attribute lists are represented in the Authorization C APIs by the datatype azn_attrlist_h_t.

An object of this class simply contains the handle to an attribute list and is used as either an input or output parameter for the methods that create, use, modify or delete an attribute list.

Several Authorization API methods take AznAttrList objects as input parameters or return AznAttrList objects as output parameters. Use AznAttrList objects to pass attribute lists between the Authorization API and the calling application.

Attribute lists are lists of name and value pairs. AznAttrList objects contain handles to the lists of name and value pairs.

Use Azn methods to add or retrieve name and value pairs from attribute lists. The values can be stored as either strings (AznString objects) or buffers (AznBuffer objects). A name can have more than one value.

Some names are defined by the Authorization API. You can also define additional names as needed by your application.

The Azn class provides methods to create attribute lists, set or get list entries, and delete attribute lists. The following table summarizes the methods that operate on attribute lists:

Task	Description
Create an attribute list	Use "Azn.attrlist_create" on page 47 to complete the following tasks: • Allocate a new, empty attribute list. • Associate a handle with the attribute list. • Return an AznAttrlist object, set with the handle.
Set an entry in an attribute list	Use "Azn.attrlist_add_entry" on page 46 to add a string name-value pair. Use "Azn.attrlist_add_entry_buffer" on page 46 to add a buffer name-value pair (AznBuffer object).
Get attribute names from an attribute list	Use "Azn.attrlist_get_names" on page 49 to get all the names in an attribute list. The names are returned as an array of strings in an AznStrings object.
Get the number of values for a specified attribute name	Use "Azn.attrlist_name_get_num" on page 50 to get the number, as an integer, of the value attributes for a specified name in the attribute list.

Task	Description
Get a value	Use "Azn.attrlist_get_entry_string_value" on page 49 to get the value attribute of a string for a specified name in an attribute list. Use "Azn.attrlist_get_entry_buffer_value" on page 48 to get the value attribute of a buffer (AznBuffer object) for a specified name in an attribute list. The specified name can have multiple values. You specify the needed value by supplying an index (integer) into the list of values.
Delete an attribute list	Use "Azn.attrlist_delete" on page 47 to delete the attribute list associated with a specified attribute list handle.

Class com.ibm.pd.Authzn.AznCreds

The AznCreds class implements an authorization credentials. The authorization credentials is represented in the Authorization C APIs by the data type azn_creds_h_t.

An object of this class simply contains the handle to a credentials structure. An AznCreds object is used as either an input or output parameter for the methods that create or use authorization credentials.

Credential handles

A credential handle refers to a credentials chain consisting of the credentials of the initiator and a series of (zero or more) intermediaries through which the initiator's request has passed.

Several Azn methods use AznCreds objects, containing credentials handles, as input parameters or output parameters. Use AznCreds objects to pass credential handles between the Authorization API and the calling application.

Variables of type AznCreds.handle are opaque handles to credential structures that are internal the Policy Director security framework.

Use the method "Azn.creds_create" on page 51 to complete the following tasks:

- Allocate a new, empty credential structure.
- Associate a handle with the credential structure.
- Return an AznCreds object, set with the handle.

Call the method "Azn.creds_delete" on page 51 to release the memory allocated for the credential structure.

Class com.ibm.pd.Authzn.AznAuthInfo

The AznAuthInfo class implements the access control information that is passed as input to the Azn.id_get_creds method within the *mechanism_info* parameter.

Objects of this class represent one of the data structures used by the Authorization C APIs for the following data types:

C API Data Type	Usage
azn_authdce_t	For DCE credentials
azn_authldap_t	For LDAP credentials
azn_unauth_t	For unauthenticated credentials

Status codes and error handling

Azn methods return an integer status code. The return value for successful completion of the method is Azn.S_COMPLETE, which is defined to be 0.

The returned status code includes both major and minor error codes.

Use "Azn.error_major" on page 58 to extract major error codes from the returned status. Major error codes are defined according to the The Open Group Authorization API Standard.

Use "Azn.error_minor" on page 58 to extract minor error codes from the returned status. The minor codes contain error messages from the Azn utility method extensions to the API, and contain error messages from the Policy Director authorization server.

Use "Azn.error_minor_get_string" on page 59 to obtain string values for the minor error codes returned by Azn.error_minor.

The list of error codes are documented in the file com.ibm.pd.Authzn.Azn.html, which is contained in the docs directory of the Policy Director Authorization Java API distribution.

Chapter 3. Using the Authorization API

This chapter includes:

Summarizing Authorization API tasks

The primary task of the Authorization API is to obtain an authorization decision from the Policy Director Authorization Service.

Use the Authorization API to present information about the user, operation, and requested resource to the Policy Director Authorization Service. Then use the Authorization API to receive the authorization decision. Your application is responsible for enforcing the decision, as appropriate.

Required tasks

To obtain an authorization decision, you must accomplish certain tasks. The following sections in this document provide a step-by-step guide to completing each of these required tasks:

Optional tasks

The Authorization API Azn class also provides methods for performing optional tasks on user credentials. The following section describes the supported optional tasks:

Runtime environment

To determine whether your network environment is configured correctly to support your application, review the following section:

Initializing the authorization service

To use the Policy Director Authorization API, an application must initialize the API. Initialization consists of specifying initialization data and calling an initialization method.

The Authorization API initialization method Azn.initialize takes as an input parameter an attribute list named init_data. To specify initialization data, you must add the necessary attributes to init_data.

Complete the instructions in the following sections:

- "Specifying the type of cache mode" on page 16
- "Adding attributes for remote cache mode" on page 17
- "Adding attributes for local cache mode" on page 17
- "Adding attributes for LDAP access" on page 20
- "Starting the authorization service" on page 21

Specifying the type of cache mode

The cache mode determines if the Authorization API talks to a Policy Director Authorization server running in the same process space (local cache mode) or in a different process space (remote cache mode) in the secure domain.

Local cache mode can increase application performance because authorization checks can be performed on the same system as the application. Local cache mode, however, requires additional configuration and maintenance of a replicated authorization database.

- For more information on remote cache mode, see “Remote cache mode” in Chapter 3 of the *Policy Director Administration Guide.*
- For more information on local cache mode, see “Local cache mode” in Chapter 3 of the *Policy Director Administration Guide*.

To specify the type of cache mode, complete the following steps:

1. Call "Azn.attrlist_create" on page 47 to obtain a handle to a new attribute list called initdata. Azn.attrlist_create takes an AznAttrList object as input, and initializes it with the handle to the attribute list.
2. Use "Azn.attrlist_add_entry_buffer" on page 46 to add the attribute Azn.init_mode and assign it a value:

Attribute	Value	Description
Azn.init_mode	local	The Policy Director Authorization Service runs in the same server process as the application using the Authorization API.
	remote	The Policy Director Authorization Service runs as a different server process from the application using the Authorization API.

Continue to the appropriate section:

- "Adding attributes for remote cache mode" on page 17.
- "Adding attributes for local cache mode" on page 17.

Adding attributes for remote cache mode

If you specified remote cache mode, use "Azn.attrlist_add_entry" on page 46 to add the attribute Azn.init_qop and assign it a value:

Attribute	Value	Description
Azn.init_qop	none	No protection.
	integrity	Data stream integrity. The data can be seen but not modified or replayed by a third party.
	privacy	Data stream privacy. The data cannot be seen, modified, or replayed by a third party.

For example, the following code shows the creation of a new attribute list. It also shows the assigning of name-value pairs for cache mode (Azn.init_mode) and quality of protection (Azn.init_qop):

```
/*** Don't use a local replica, use the authorization server ***/
status = Azn.attrlist_add_entry(initdata,
                                Azn.init_mode,
                                "remote");

if ( status != Azn.S_COMPLETE ) return status;

/*
 * Set quality of protection for communications with ivacld
 * to be privacy (encrypted).
 */
status = Azn.attrlist_add_entry(initdata,
                                Azn.init_qop,
                                "privacy");

if ( status != Azn.S_COMPLETE ) return status;
```

Initialization of remote cache mode is now complete.

- If your secure domain uses an LDAP user registry, refer to "Adding attributes for LDAP access" on page 20.
- If your secure domain uses a DCE user registry, refer to "Starting the authorization service" on page 21.

Adding attributes for local cache mode

When you specify local cache mode, you must decide how the local copy of the authorization database will be updated.

Choose one of the following methods to implement updating:

- Set the Authorization API to poll the master authorization service database.
- Register the local (replicated) database with the master database, and enable a listener process on the local database's system. This process listens for update notifications.
- Configure the Authorization API to both poll and listen.

- Configure the Authorization API to neither poll nor listen. This could be useful, for example, when the local system is not connected to a network.

The above methods are configured by adding attributes to the init_data attribute list.

Complete all the steps in this section in order to implement your chosen method:

1. Use Azn.attrlist_add_entry to specify pathnames for files used by the authorization service.

Attribute	**Value**	**Description**
Azn.init_db_file	*filename*	Path name to the persistent authorization policy database replica.
Azn.init_audit_file	*filename*	Path and file name for the file that collects Authorization API audit events.

2. Use Azn.attrlist_add_entry to configure the Authorization API to poll the master authorization database.

Attribute	**Value**	**Description**
Azn.init_cache_refresh_interval		
	disable	Refreshing of the local authorization policy database disabled.
	default	600 seconds.
	number of seconds	Number of seconds between refreshes of the local authorization policy database. Set appropriate values to ensure that the replicated database is updated in a timely manner to reflect changes made to the master database.

3. Use Azn.attrlist_add_entry to configure the notification listener.

Attribute	**Value**	**Description**
Azn.init_listen_flags	disable	Disable the notification listener.
	enable	Enable the notification listener.
	When you select enable, you can also specify any combination of the following values. The values are logically OR'd together.	
	use_tcp_port	Enable the listener to use Transmission Control Protocol (TCP).
	use_udp_port	Enable the listener to use User Datagram Protocol (UDP).
	dynamic_port_selection	Instruct the listener to use randomly assigned ports.

4. If you enable the notification listener, you must use the **ivadmin** command to inform the Policy Director Management server (**ivmgrd**) of your location in order to receive notification of updates. Use the **ivadmin server register dbreplica** command to inform the Policy Director Authorization Service (specifically, the Management server) of the existence and location of applications using the Authorization API in local cache mode.

   The following syntax applies:

   ```
   ivadmin>server register dbreplica server-name ns-location server-host
   ```

   Where:

Option	Description
server-name	A name (or label) for this application. This is the name that appears in the display of the object space on the Management Console and in the ivadmin server list command.
ns-location	The RPC entry in the CDS namespace where the application exports its RPC bindings.
server-principal	The name of the DCE principal representing this application process.
server-host	The Domain Name System (DNS) name or IP address of the machine where this application process resides.

5. If you enabled the notification listener, use Azn.attrlist_add_entry to add the following attributes:

   Note: If you disabled the notification listener, skip this step.

Attribute	Value	Description
Azn.init_tcp_port	*port number*	If you specified use_tcp_port and did not specify dynamic_port_selection for the attribute Azn.init_listen_flags, use this value to specify a TCP port.
Azn.init_udp_port	*port number*	If you specified use_udp_port and did not specify dynamic_port_selection for the attribute Azn.init_listen_flags, use this value to specify a UDP port.
Azn.init_namespace_location	*CDS location*	Specify the CDS namespace location for exporting the RPC endpoints for local policy cache updates.

For example, the following code shows the creation of a new attribute list init_data, and also shows the addition of entries to specify configuration settings for local cache mode:

```
Azn.attrlist_create(init_data);

/*** Use a local DB replica ***/
status = Azn.attrlist_add_entry(initdata,
                                Azn.init_mode,
                                "local");
if (status != Azn.S_COMPLETE)return (status);
```

```
/*** The file name of the replica policy database ***/
status = Azn.attrlist_add_entry(initdata,
                                Azn.init_db_file,
                                "./auth_demo.db");
if (status != Azn.S_COMPLETE)return (status);

/*** The file name of the audit file ***/
status = Azn.attrlist_add_entry(initdata,
                                Azn.init_audit_file,
                                "./auth_demo.audit");
if (status != Azn.S_COMPLETE)return (status);

/*** Enable polled updates at the default interval ***/
status = Azn.attrlist_add_entry(initdata,
                                Azn.init_cache_refresh_interval,
                                "default");
if (status != Azn.S_COMPLETE)return (status);

/*** Enable the update notification listener ***/

status = Azn.attrlist_add_entry(initdata,
                                Azn.init_listen_flags,
                                "enable");
if (status != Azn.S_COMPLETE)return (status);

/*** Enable TCP port ****/
status = Azn.attrlist_add_entry(initdata,
                                Azn.init_listen_flags,
                                "use_tcp_port");
if (status != Azn.S_COMPLETE)return (status);

/*** Set TCP port number ****/
status = Azn.attrlist_add_entry(initdata,
                                Azn.init_tcp_port,
                                "6056");
if (status != Azn.S_COMPLETE) return (status);

/*** Set CDS location ****/
status = Azn.attrlist_add_entry(initdata,
                                Azn.init_namespace_location,
                                CDSloc);
if (status != Azn.S_COMPLETE)return (status);
```

Adding attributes for LDAP access

When your application runs in a Policy Director secure domain that uses an LDAP user registry, you must provide the LDAP configuration settings to the Authorization API. The required LDAP configuration settings match the settings that were entered when Policy Director was installed on the local system.

Note: When your application runs in a Policy Director secure domain that uses a DCE user registry, skip this step and go to"Starting the authorization service" on page 21.

1. Use Azn.attrlist_add_entry to add the following attributes to the init_data attribute list:

Attribute	Value	Description
Azn.init_ldap_host	*host name*	Host name of LDAP server.
Azn.init_ldap_port	*port number*	Port number for communicating with the LDAP server.

Attribute	Value	Description
Azn.init_ldap_admin_dn	*LDAP DN*	Distinguished Name of the LDAP administrator.
Azn.init_ldap_admin_pwd	*password*	Password for the LDAP administrator.

2. If the communication between the Policy Director Authorization server and the LDAP server is over Secure Sockets Layer (SSL), use Azn.attrlist_add_entry to add the following attributes to the init_data attribute list:

Attribute	Value	Description
Azn.init_ldap_ssl_keyfile	*filename*	Name of the SSL key file.
Azn.init_ldap_ssl_keyfile_dn	*KeyLabel*	Key label to identify the client certificate that is presented to the LDAP server.
Azn.init_ldap_ssl_keyfile_pwd	*password*	Password to access the SSL key file.

Starting the authorization service

Complete the following steps:

1. Ensure that the attribute list initdata has been created and filled in, as described in the preceding sections.
2. Call Azn.initialize to bind to and initialize the authorization service.

   For example:

```
/* Start the service */
status = Azn.initialize(initdata, initinfo);
if (status != Azn.S_COMPLETE) return(status);
```

In the example code above, Azn.initialize returns the attribute list initinfo. This attribute list is appended with any initialization information attributes that apply. This includes the Azn.C_VERSION attribute, which contains the version number of the API implementation.

Note: To re-initialize the API, use Azn.shutdown and then call Azn.initialize.

For more information, see "Azn.initialize" on page 60.

Authenticating an API application

The API application must establish its own authenticated identity within the Policy Director secure domain, in order to request authorization decisions from the Policy Director Authorization Service.

Before you run the Authorization API application for the first time, you must create a unique identity for the application in the Policy Director secure domain.

In order for the authenticated identity to perform API checks, the application must be a member of at least one of the following groups:

- **ivacld-servers**

  This group membership is needed for applications using local cache mode.

- **remote-acl-users**

  This group membership is needed for applications using remote cache mode.

When the application wants to contact one of the secure domain services, it must first log in to the secure domain.

The Policy Director Authorization API provides two utility methods the application can use to log in and obtain an authenticated identity. One method performs a login by using username and password information. The other method performs a DCE login by using a keytab file.

Use the appropriate API login methods, as described in the following sections:

- **"Logging in using a DCE keytab file"** on page 22
- **"Logging in using a password"** on page 22

Logging in using a DCE keytab file

Some application servers are executed non-interactively, such as in response to an access request from an application client. These application servers must establish an authenticated identity without manual intervention by an administrator.

To avoid the need for manual intervention, the application developer can create and store a password in a keytab file.

The Authorization API utility method Azn.util_server_authenticate submits the user name and the name of the keytab file to the Policy Director authentication service. The Policy Director authentication service can use the DCE keytab file to establish an authenticated identity.

For example, the following code logs in a server svrPrin using a keytab file svrKeytab:

```
status = Azn.util_server_authenticate(svrPrin, svrKeytab);
if ( status != Azn.S_COMPLETE ) {

{
   System.out.println("\nCould not perform keytab login.\n");
   System.exit(1);
}
```

Note: You can use Azn.util_server_authenticate in a Policy Director secure domain that uses an LDAP user registry, but it can only be used for DCE principals (as registered in a DCE user registry).

For more information, see "Azn.util_server_authenticate" on page 64.

Logging in using a password

Some applications might be used by more than one identity in the Policy Director secure domain. These applications can choose their login identity based on application requirements. For example, the application can prompt the user, or examine user information contained in an HTTP header, or simply supply a username and password that denotes a category of user.

The Authorization API provides the utility method Azn.util_client_authenticate to enable the application to log in as a specific identity with a user name and password.

For example, the following code logs in the application as “testuser**”**:

```
/* Login and start context refresh thread */
status = Azn.util_client_authenticate(“testuser”,“testuserpwd”);

if ( status != Azn.S_COMPLETE )
{
   System.out.println("\nCould not perform client login\n");
   System.exit(1);
}
```

You can use Azn.util_client_authenticate in a Policy Director secure domain with a DCE user registry.

For more information, see "Azn.util_client_authenticate" on page 62.

Obtaining an identity for a user

The application must determine the identity of the user who has submitted a request. The identity can be expressed as one of the following types of users:

- **Authenticated**

  In this case, the user’s identity in the secure domain is registered in either an LDAP or DCE user registry. The user is authenticated, and information about the user can be obtained. This information includes, for example, the Distinguished Name (LDAP) or principal (DCE).

- **Unauthenticated**

  In this case, the user’s identity in the secure domain is not specifically registered in either an LDAP or DCE user registry. The user is defined to be unauthenticated, and further information about the user’s identity is irrelevant to the authorization process.

Applications can obtain user identities through a variety of methods. These can include the use of a Credentials Acquisition Server, or a call to an application-specific method for querying user registries and establishing a security (login) context.

Optionally, applications can use the Policy Director Authorization API utility method Azn.util_password_authenticate to obtain user identity information from the secure domain.

The method Azn.util_password_authenticate requires the user name and password as input parameters. Typically, an application receives a user name and password from the user who initiated the access request.

The method performs a login using the supplied user name and password. If the login is successful, the method returns the following information:

- An AznString object named mechanism_id, set with the authentication mechanism (DCE or LDAP) that was used.
- An AznAuthInfo object named authinfo, set with the user identity information.

Note: The method Azn.util_password_authenticate does not obtain a security (login) context for the user.

For more information, see "Azn.util_password_authenticate" on page 63.
After the application has obtained identity information for the user, you can use the Authorization API to obtain authorization credentials for the user.

Obtaining user authorization credentials

In order to submit an authorization request to the Policy Director Authorization Service, an application must obtain authorization credentials for the user making the request. The authorization credentials contain user identity information that is needed to make authorization decisions, such as group memberships and a list of actions or rights that the user can exercise.

To obtain credentials for a user who has submitted an access request, an application must obtain user identity information from the user registry (DCE or LDAP) that is used by the Policy Director secure domain.

The Authorization API method Azn.id_get_creds takes user identity information as input parameters and returns user authorization credentials.

The credentials can then be submitted to the authorization service for an authorization decision.

Note: Identity information can also be obtained from a privilege attribute certificate (PAC). See "Converting credentials to the native format" on page 31.

To obtain a credential, complete the instructions in each of the following sections:

1. "Specifying the authorization authority" on page 24
2. "Specifying authentication user registry type" on page 24
3. "Specifying user authentication identity" on page 25
4. "Specifying additional user information" on page 25
5. "Obtaining authorization credentials for the user" on page 26

Specifying the authorization authority

Assign the appropriate value for the authorization authority to a string. This string is passed as the parameter authority to Azn.id_get_creds. Set authority to null to specify Policy Director authorization.

Specifying authentication user registry type

Applications must know the type of user registry used in the Policy Director secure domain, in order to obtain an authenticated identity for the user. The type of registry used was determined in "Obtaining an identity for a user" on page 23.

If the user was not authenticated in a user registry, then the user registry type is unauthenticated.

Assign a value for the type of user authentication identity to a string. This string is passed as the parameter mechanism_id to Azn.id_get_creds.

Set mechanism_id to one of the following values:

User Registry	Value
DCE User Registry	Azn.IV_DCE
LDAP User Registry	Azn.IV_LDAP
Unauthenticated	Azn.IV_UNAUTH

Specifying user authentication identity

For each user to be authenticated, information is loaded into the data structure that corresponds to the type of user registry used in the secure domain, or is loaded into a data structure corresponds to a user category of “unauthenticated”.

If the user is authenticated, you must load the user’s identity into a user_identity variable in an AznAuthInfo object.

User Identity Type	Variable	String	Example
DCE User Registry	AznAuthInfo.user_identity	principal	cell_admin
LDAP User Registry	AznAuthInfo.user_identity	ldap_dn	cn=root
Unauthenticated User	*none*	*none*	*none*

If the user is unauthenticated, you do not have to load an identity into AznAuthInfo.user_identity.

Specifying additional user information

When the application authenticates the user, the application can optionally obtain additional information about the user. This additional information is for use by the application as needed. The Policy Director Authorization Service does not use this information.

The application can store the additional user information as variables in an AznAuthInfo object, as described in the table below.

Variable	Description
auth_method	Indicates that the user was authenticated through either the DCE user registry or the LDAP user registry. This value can be any string that is useful to the application. Not used for unauthenticated users.
authnmech_info	Additional authentication information. This value can be any string that is useful to the application. For example, if the DCE authentication was accomplished using SSL certificates, the certificate’s Distinguished Name could be stored here. Not used for unauthenticated users.
qop	Quality of protection level for requests made by this user. This level is set by the application and is specified as an arbitrary character string.
user_info	Additional user information for auditing purposes. This string can contain any information that is useful to the application.

Variable	Description
browser_info	Information about the type of browser through which the user has submitted the request, if applicable. This string can contain any information that is useful to the application.
ipaddr	The IP address of the user. This integer is optional information for use by the application.

The AznAuthInfo object that contains all of the above user information will be passed as an input parameter to Azn.id_get_creds.

Obtaining authorization credentials for the user

To obtain authorization credentials, call Azn.id_get_creds with the following input parameters:

Parameter	Description
authority	The authorization authority, as described in "Specifying the authorization authority" on page 24.
mechanism_id	The authentication mechanism, as described in "Specifying authentication user registry type" on page 24.
mechanism_info	An AznAuthInfo object containing user information, as described in the following sections: • "Specifying user authentication identity" on page 25. • "Specifying additional user information" on page 25

The Azn.id_get_creds method returns an AznCreds object set to the authorization credentials for the user.

For example, the following sample code demonstrates the assigning of identity information for a user authenticated in an LDAP user registry, and calls Azn.id_get_creds to obtain authorization credentials:

```
AznAuthInfo      ldap_minfo = new AznAuthInfo();
String mech = null;
AznAuthInfo mech_info = null;

/*** Create new credentials object ***/
AznCreds         creds = new AznCreds();

/* Specify authentication registry type */
mech = IV_LDAP;

/* Specify LDAP user name */
ldap_minfo.user_identity = “cn=testuser”;

/* Set LDAP user information. Note: these values are just placeholders
*/
ldap_minfo.auth_method = "ldap_auth_method";
ldap_minfo.authnmech_info = "ldap_authnmech_info";
ldap_minfo.qop = "ldap_qop";
ldap_minfo.user_info = "ldap_user_info";
ldap_minfo.browser_info = "ldap_browser_info";
ldap_minfo.ipaddr = 0x0a000002;

mech_info = ldap_info;

/* Obtain an authorization credential. Specify the authority as NULL */
```

```
status = Azn.id_get_creds(null, mech, mech_info, creds);
if (status != Azn.S_COMPLETE)
{
    System.out.println("Could not get creds.");
    continue;
}
```

For more information, see "Azn.id_get_creds" on page 59. Refer also to the Authorization API demonstration program. See "Running the example program AznDemo" on page 33.

The application is now ready to submit the authorization request. See "Obtaining an authorization decision" on page 27.

Obtaining an authorization decision

After the application has obtained authorization credentials for the user, the application passes the requested operation and the requested resource to the Authorization API method Azn.decision_access_allowed. This method returns the authorization decision.

To obtain an authorization decision, complete the instructions in each of the following sections:

Mapping the user operation to a Policy Director permission

The operation requested by the user must correspond to one of the operations for which a Policy Director permission has been defined. The operation is a standard action supported in all Policy Director secure domains. Examples operations are Azn.operation_read and Azn.operation_traverse.

Note: For a complete list of supported operations, see "Class com.ibm.pd.Authzn.Azn" on page 36.

Alternatively, the operation can be a custom operation defined by an external authorization service.

Pass the operation as a string to Azn.decision_access_allowed.

Mapping the requested resource to a protected object

The requested resource to query for must correspond to a resource that has been defined as a protected object in the secure domain's protected object namespace.

The resource can be a standard WebSEAL protected resource, such as a file in the Web space. Alternatively, the resource can be a custom protected object.

The following code places a value for a resource named "/example-object" into an AznString object.

Pass the requested resource as a string to Azn.decision_access_allowed.

Assigning the user credentials to a credentials handle

The authorization credentials for a user obtained in "Obtaining user authorization credentials" on page 24 can be accessed through the AznCreds object returned by Azn.id_get_creds.

These credentials contain the user's identity information and include information such as the user's group membership and permitted operations.

Pass the AznCreds object as an input parameter to Azn.decision_access_allowed.

Note: Authorization credentials can also be obtained from Azn.pac_get_creds. See "Converting credentials to the native format" on page 31.

Building an attribute list for additional application information

The Policy Director Authorization API provides the extended method Azn.decision_access_allowed_ext for obtaining an access decision. This method extends Azn.decision_access_allowed by providing an additional input parameter and an additional output parameter.

These parameters can be used to supply additional information as needed by the application. The Policy Director Authorization Service does not use these parameters when making the access control decision. However, you can write external authorization servers to use this information.

The parameters consist of an attribute list. You can build an attribute list of any length to hold information specific to the application.

To add additional application-specific context, complete the following steps:

1. Use Azn.attrlist_create to create a new, empty attribute list named app_context.
2. Use Azn.attrlist_add_entry or Azn.attrlist_add_entry_buffer to add attributes.
3. When all attributes have been added, pass app_context as an input parameter to Azn.decision_access_allowed_ext.

For more information, see "Azn.decision_access_allowed_ext" on page 57.

Obtaining an authorization decision

To obtain an authorization decision, call one of the following methods:

- Azn.decision_access_allowed
- Azn.decision_access_allowed_ext

For example:

```
AznInteger permitted;

/* Perform authorization check */
```

```
status = Azn.decision_access_allowed(creds,
                                     obj_name.value,
                                     operation.value,
                                     permitted);

if ( status == Azn.S_COMPLETE )
{
    System.out.print("\n\nResult: ");
    if ( permitted.value == Azn.PERMITTED )
    {
        System.out.println("Permitted.\n");
    }
    else
    {
        System.out.println("Not permitted.\n");
    }
}
```

If the API is operating in remote cache mode, the authorization request will be forwarded to the Policy Director Authorization server (**ivacld**). The Authorization Server makes the decision and returns the result.

If the API is operating in local cache mode, the API uses the local authorization policy database replica to make the authorization decision.

The result of the access request is returned in the following object::

Type	Parameter	Description
AznInteger	*permission*	The result of the access request. Consists of one of the following constants: Azn.C_PERMITTED Azn.C_NOT_PERMITTED

The extended method Azn.decision_access_allowed_ext also returns the following information:

Type	Parameter	Description
AznAttrList	*permission_info*	Application-specific context information contained in attribute list.

For more information on the above methods, see:

- "Azn.decision_access_allowed" on page 56
- "Azn.decision_access_allowed_ext" on page 57

Cleaning up and shutting down

Releasing allocated memory

The Authorization API provides the following methods to perform the releasing of memory:

- "Azn.attrlist_delete" on page 47

  Use this method to delete the attribute list referenced by the handle contained in AznAttrList objects. This releases memory allocated by the corresponding Authorization C API.

- "Azn.creds_delete" on page 51

  Use this method to delete the credentials referenced by the handle contained in AznCreds objects. This releases memory allocated by the corresponding Authorization C API.

Shutting down the Authorization Api

When an application has obtained an authorization decision and when it does not need further authorization decisions, use "Azn.shutdown" on page 62 to disconnect from and shut down the Authorization API.

For example:

```
status = Azn.shutdown();
```

Handling credentials (optional tasks)

The Authorization API provides methods to accomplish the following optional tasks:

- "Converting credentials to a transportable format" on page 30
- "Converting credentials to the native format" on page 31
- "Creating a chain of credentials" on page 31
- "Determining the number of credentials in a credentials chain" on page 31
- "Obtaining a credential from a chain of credentials" on page 31
- "Modifying the contents of a credential" on page 32
- "Obtaining an attribute list from a credential" on page 32

Converting credentials to a transportable format

Use the method "Azn.creds_get_pac" on page 53 to place user credentials into a format that can be transported across a network to another application. Use this method when you need to delegate the authorization decision to an application on another system.

Complete the following steps:

1. Set the input string pac_svc_id to null.
2. Set the input credentials AznCreds object creds to the AznCreds object returned by a previous call to Azn.id_get_creds or Azn.pac_get_creds.
3. Call Azn.creds_get_pac.

The privilege attribute certificate (PAC) is returned in an AznBuffer object named pac. This buffer can be transported to another system, where the method Azn.pac_get_creds can be used to return the credentials to a native format.

Converting credentials to the native format

Use the method "Azn.pac_get_creds" on page 61 when an application receives credentials from another system on the network. Typically, these credentials are placed into a buffer by Azn.creds_get_pac.

Complete the following steps:

1. Set the input parameter string pac_svc_id to null.
2. Set the input parameter AznBuffer pac to the AznBuffer object returned by a previous call to Azn.creds_get_pac.
3. Call Azn.pac_get_creds.

This method returns an AznCreds object, for access by other Authorization API methods.

Creating a chain of credentials

Use the method "Azn.creds_combine" on page 50 to combine, or chain, two credentials together. Use this, for example, when the credentials for a server application must be combined with user credentials in order to delegate the authorization decision to another application.

Complete the following steps:

1. Set an AznCreds object “creds” with the credentials of the initiator of the request.
2. Set an AznCreds object “creds_to_add” with the credentials to be added.
3. Call Azn.creds_create to create a new, empty credentials structure.
4. Call Azn.creds_combine.

The combined credentials are placed in a credentials structure that can be referenced by the AznCreds object named ‘combined_creds”.

Determining the number of credentials in a credentials chain

Use the method "Azn.creds_num_of_subjects" on page 55 to determine the number of credentials that are contained in a credentials chain. Credentials chains are created by the Azn.creds_combine method.

This methods takes as an input parameter an AznCreds object set with the credentials chain, and returns an AznInteger object containing the number of credentials.

Obtaining a credential from a chain of credentials

Use the method "Azn.creds_for_subject" on page 52 to extract individual credentials from a credentials chain. Credentials chains are created by the Azn.creds_combine method.

Complete the following steps:

1. Set an AznCreds object “creds” with the credentials chain.
2. Set an AznInteger object “subject_index” with the index number of the needed credential within the credentials chain.

   The credentials of the user who made the request are always stored at index 0. To retrieve the credentials for the initiator (user), you can pass the constant Azn.C_INITIATOR_INDEX as the value for subject_index.

Use Azn.creds_num_of_subjects, if necessary, to determine the number of credentials in the chain.

3. Call Azn.creds_for_subject.

This method returns the requested credentials in the AznCreds object *new_creds*.

Modifying the contents of a credential

Use the method "Azn.creds_modify" on page 54 to modify a credential by placing additional information, which is contained in an attribute list, into the credentials structure. Use this method when you need to add application-specific information to a user's credentials.

Complete the following steps:

1. Use the AznAttrList methods to create an attribute list containing the information to be added. Set an AznAttrList object named "mod_info" to the new attribute list.

   For more information on attribute lists, see "Class com.ibm.pd.Authzn.AznAttrList" on page 11.

2. Set the credential modification service string "mod_svc_id" to null.
3. Set an AznCreds object "creds" to the credentials to be modified.
4. Call Azn.creds_create to create a new, empty credentials structure.
5. Call Azn.creds_modify.

The modified credentials are placed in the AznCreds object *new_creds*.

Obtaining an attribute list from a credential

Use the method "Azn.creds_get_attrlist_for_subject" on page 53 to obtain information, in the form of an attribute list, from a credential. Attribute lists are added to credentials structures by calls to Azn.creds_modify**.**

You can use this method to obtain the attribute list for a credential that is part of a credentials chain.

Complete the following steps:

1. Set an AznCreds object named "creds" to the credentials chain.
2. Set an AznInteger object named "subject_index" to the index of the credential within the credentials chain.

   If the credential is not part of a chain, set subject_index to 0.

   The credentials of the user who made the request are always stored at index 0. To retrieve the credentials for the initiator (user), you can pass the constant Azn.C_INITIATOR_INDEX as the value for subject_index.

   Use Azn.creds_num_of_subjects, if necessary, to determine the number of credentials in the chain.

3. Call Azn.attrlist_create to create a new, empty attribute list.
4. Call Azn.creds_get_attrlist_for_subject.

The method returns an AznAttrList object named "creds_attrlist" containing the credential's attribute information.

Chapter 4. Deploying applications with the Authorization API

To deploy an application with the Authorization API, verify that your environment contains the necessary supporting software. You can test your environment by building and running the example program that is provided with the Authorization API.

See the following sections:

- "Software requirements" on page 33
- "Running the example program AznDemo" on page 33

Software requirements

1. Install a Java runtime environment in order to run Java programs.
2. Applications that have been developed with the Policy Director Authorization Java API must be run on systems that are configured into a Policy Director secure domain. The minimum Policy Director installation on a system that will run an application is:
   - Policy Director Base (IVBase)
   - Policy Director Authorization server (IVAcld)
   - Policy Director Application Development Kit (IVAuthADK)

   Note: When the Policy Director secure domain uses an LDAP user registry, the application deployment system must have an LDAP client installed.

The application runtime environment must include a DCE client runtime. The DCE runtime is installed as a prerequisite to the Policy Director installations described above.

Note: On Windows NT, Policy Director NetSEAT client provides the DCE client runtime environment.

Running the example program AznDemo

The Policy Director Authorization API is provided with an example program called **AznDemo** that demonstrates use of the Authorization Java API.

See the README file for instructions on running the AznDemo program. The README file is located in the same installation directory as the AznDemo program.

Chapter 5. Authorization API: Java Reference

The Java implementation of the Authorization API consists of objects that are extensions to class java.lang.Object.

This section contains a reference section for each of the following classes:

Class com.ibm.pd.Authzn.Azn

```
public class Azn extends Object
```

Description

The Azn class implements static native methods used to invoke the Policy Director Authorization APIs which are C based APIs. The C APIs are fully documented in the Policy Director Programming Guide and Reference. There is a one-to-one mapping between the Java methods implemented by this class and the C based Authorization APIs.

The C based APIs all begin with azn_ whereas the methods in this class are named by what follows the azn_ in the C API function name. For example, the C API azn_initialize function corresponds to the initialize method in this class and since it is a static method it is invoked using the class name Azn.initialize.

The parameters to the methods in this class correspond as closely as possible to the parameters for the C APIs.

Variable Index

Constructor Index

Method Index

Variables

IV_UNAUTH

Syntax

```
public static String IV_UNAUTH
```

Remarks

Mechanism ID for an unauthenticated user which is passed to the Azn.id_get_cred method to indicate the type of AznAuthInfo object that also passed to id_get_cred.

IV_DCE

Syntax

```
public static String IV_DCE
```

Remarks

Mechanism ID for a DCE authenticated user which is passed to the id_get_cred method to indicate the type of AznAuthInfo object that also passed to id_get_cred.

IV_LDAP

Syntax

```
public static String IV_LDAP
```

Remarks

Mechanism ID for an LDAP authenticated user which is passed to the id_get_cred method to indicate the type of AznAuthInfo object that also passed to id_get_cred.

operation_attach

Syntax

```
public static String operation_attach
```

Remarks

Operation string for attach.

Operation string declarations are parameters to the Azn.decision_access_allowed and Azn.decision_access_allowed_ext methods. The actual string values of these operations are an internal implementation detail and should not be relied upon.

The operations can be concatenated together to form complex operation strings. For example, to request a read / modify operation, concatenate the strings operation_read and operation_modify.

operation_browse

Syntax

```
public static String operation_browse
```

Remarks

Operation string for browse.

operation_control

Syntax

```
public static String operation_control
```

Remarks

Operation string for control.

operation_traverse

```
public static String operation_traverse
```

Remarks

Operation string for traverse.

operation_delegation

Syntax

```
public static String operation_delegation
```

Remarks

Operation string for delegation.

operation_view

Syntax

```
public static String operation_view
```

Remarks

Operation string for view.

operation_modify

Syntax

```
public static String operation_modify
```

Remarks

Operation string for modify.

operation_delete

Syntax

```
public static String operation_delete
```

Remarks

Operation string for delete.

operation_server_admin

Syntax

```
public static String operation_server_admin
```

Remarks

Operation string for server admin.

operation_audit

Syntax

```
public static String operation_audit
```

Remarks

Operation string for audit.

operation_integrity

Syntax

```
public static String operation_integrity
```

Remarks

Operation string for integrity.

operation_privacy

Syntax

```
public static String operation_privacy
```

Remarks

Operation string for privacy.

operation_read

Syntax

```
public static String operation_read
```

Remarks

Operation string for read.

operation_execute

Syntax

```
public static String operation_execute
```

Remarks

Operation string for execute.

operation_list_directory

Syntax

```
public static String operation_list_directory
```

Remarks

Operation string for list directory.

operation_connect

Syntax

```
public static String operation_connect
```

Remarks

Operation string for connect.

operation_forward

Syntax

```
public static String operation_forward
```

Remarks

Operation string for forward.

init_mode

Syntax

```
public static String init_mode
```

Remarks

Attribute name for the Authorization API initialization mode. Values are:

- local

  Specifies the use of policy cache replica of the authorization database.

- remote

  Specifies communication with the master copy of the authorization database, by making RPC requests to the Policy Director Authorization Server.

init_qop

Syntax

```
public static String init_qop
```

Remarks

Attribute name for the quality of protection for communications with IVAcld. Values are "none", "integrity" or "privacy".

init_db_file

Syntax

```
public static String init_db_file
```

Remarks

Attribute name for the path and filename used to contain cached authorization policy.

init_audit_file

Syntax

```
public static String init_audit_file
```

Remarks

Attribute name for the audit path and filename to collect Authorization API audit events.

init_cache_refresh_interval

Syntax

```
public static String init_cache_refresh_interval
```

Remarks

Attribute name of the interval in seconds for local policy cache polled updates. Values can be "disable", "default" or a string time in seconds.

init_listen_flags

Syntax

```
public static String init_listen_flags
```

Remarks

Attribute name for flags to enable the reception of policy cache update notifications. Values can be a combination of: "disable", "enable", "use_tcp_port", "use_udp_port" and "dynamic_port_selection". Multiple values are accepted for this attribute name and are logically OR'd together. A "disable" value overrides all others and disables the notification listener.

init_namespace_location

Syntax

```
public static String init_namespace_location
```

Remarks

Attribute name for the CDS namespace location for exporting the RPC endpoints for local policy cache updates.

init_tcp_port

Syntax

```
public static String init_tcp_port
```

Remarks

Attribute name for the TCP port upon which policy cache updates are received.

init_udp_port

Syntax

```
public static String init_udp_port
```

Remarks

Attribute name for the UDP port upon which policy cache updates are received.

init_ldap_host

Syntax

```
public static String init_ldap_host
```

Remarks

Attribute name for the LDAP server host name.

init_ldap_port

Syntax

```
public static String init_ldap_port
```

Remarks

Attribute name for the LDAP server host port (a numerical string).

init_ldap_admin_dn

Syntax

```
public static String init_ldap_admin_dn
```

Remarks

Attribute name for the LDAP server administrator's distinguished name.

init_ldap_admin_pwd

Syntax

```
public static String init_ldap_admin_pwd
```

Remarks

Attribute name for the LDAP server administrator's password.

init_ldap_ssl_keyfile

Syntax

```
public static String init_ldap_ssl_keyfile
```

Remarks

Attribute name for the LDAP server's SSL keyfile.

init_ldap_ssl_keyfile_dn

Syntax

```
public static String init_ldap_ssl_keyfile_dn
```

Remarks

Attribute name for the LDAP server's SSL keyfile distinguished name.

init_ldap_ssl_keyfile_pwd

Syntax

```
public static String init_ldap_ssl_keyfile_pwd
```

Remarks

Attribute name for the LDAP server's SSL keyfile password.

Constructor

Azn

Syntax

```
Azn()
```

Remarks

Constructor

Methods

attrlist_add_entry

Adds a name or string-value entry to an attribute list.

Syntax

```
public static native int attrlist_add_entry(AznAttrList attr_list,
                                             String attr_name,
                                             String string_value)
```

Parameters

attr_list **- input**
AznAttrList object.

attr_name **- input**
Name attribute of the entry to be added.

string_value **- input**
Value (string) attribute of the entry to be added.

Remarks

This method adds an entry to the attribute list *attr_list*. The added entry will have name *attr_name* and value *string_value.*

This call can be issued multiple times with the same *attr_list* and the same *attr_name* but with different string values. When this is done, *attr_list* contains multiple values for the specified name.

The *attr_name* and *string_value* input parameters are copied into a new attribute list entry.

Return Values

Status return code which can be passed to the error_major and error_minor methods to retrieve the Azn major and minor error code values.

attrlist_add_entry_buffer

Adds a name/buffer value entry to an attribute list.

Syntax

```
public static native int attrlist_add_entry_buffer(
                                AznAttrList attr_list,
                                String attr_name,
                                AznBuffer buffer_value)
```

Parameters

attr_list **- input**
AznAttrList object.

attr_name **- input**
Name attribute of the entry to be added.

buffer_value **- input**
AznBuffer object with the binary value for the new attribute.

Remarks

This method adds an entry to the attribute list, *attr_list*. The added entry will have name *attr_name* and value *buffer_value*.

This method can be issued multiple times with the same *attr_list* and the same *attr_name*, but with different *buffer_values*. When this is done, *attr_list* contains multiple values for the specified name.

The *attr_name* and *buffer_value* input parameters are copied into a new attribute list entry.

Return Values

Status return code which can be passed to the error_major and error_minor methods to retrieve the Azn major and minor error code values.

attrlist_create

Creates an attribute list.

Syntax

```
public static native int attrlist_create(AznAttrList attr_list)
```

Parameters

attr_list **- input /output**
AznAttrList object.

Remarks

This method creates a new and empty attribute list. Pass a new AznAttrList object *attr_list* as the input parameter. The *attr_list* object is also an output parameter.

Return Values

Status return code which can be passed to the error_major and error_minor methods to retrieve the Azn major and minor error code values.

attrlist_delete

Deletes an attribute list.

Syntax

```
public static native int attrlist_delete(AznAttrList attr_list)
```

Parameters

attr_list **- input /output**
AznAttrList object.

Remarks

This method deletes an attribute list. The *attr_list* object passed to this method is both an input and output parameter. The attribute names and values in the attribute list are released. The *attr_list* object is set to an invalid attribute list.

Return Values

Status return code which can be passed to the error_major and error_minor methods to retrieve the Azn major and minor error code values.

attrlist_get_entry_buffer_value

Returns a single specified-value attribute for a name attribute that has multiple values that are contained in buffers.

Syntax

```
public static native int attrlist_get_entry_buffer_value(
                                                AznAttrList attr_list,
                                                String attr_name,
                                                int value_index,
                                                AznBuffer buffer_value)
```

Parameters

attr_list **- input**
AznAttrList object.

attr_name **- input**
Name attribute of the entry from which the value attribute is to be returned.

value_index **- input**
Index within the entry of the string attribute value o be returned.

buffer_value **- input /output**
AznBuffer object for the returned string attribute value.

Remarks

This method returns a binary value associated with an attribute name in the specified attribute list. The returned value attribute is the one at position *value_index* within the entry whose name attribute is specified by *attr_name*.

The *value_index* parameter is the index within the attribute entry for the specified binary value. The first value attribute for any particular name attribute within an attribute list has index 0.

The buffer_value object is both an input and output parameter. The returned binary value is set in this object.

Return Values

Status return code which can be passed to the error_major and error_minor methods to retrieve the Azn major and minor error code values.

attrlist_get_entry_string_value

Returns a single specified value attribute for a name attribute that has multiple values that are strings.

Syntax

```
public static native int attrlist_get_entry_string_value(
                                                    AznAttrList attr_list,
                                                    String attr_name,
                                                    int value_index,
                                                    AznString string_value)
```

Parameters

attr_list **- input**
AznAttrList object.

attr_name **- input**
Name attribute of the entry from which the value attribute is to be returned.

value_index **- input**
Index within the entry of the string attribute value o be returned.

string_value **- input /output**
AznString object for the returned string attribute value.

Remarks

This method returns one string-type value attribute in *string_value*. The returned value attribute is the one at position *value_index* within the set of value attributes belonging to the name attribute that is specified by *attr_name*.The first value attribute for a specified name attribute within an attribute list has index 0.

The *string_value* object is both an input and output parameter. The returned string value will be set in this object.

Return Values

Status return code which can be passed to the error_major and error_minor methods to retrieve the Azn major and minor error code values.

attrlist_get_names

Returns the list of all name attributes appearing in entries of the attribute list.

Syntax

```
public static native int attrlist_get_names(AznAttrList attr_list,
                                            AznStrings attr_names)
```

Parameters

attr_list **- input**
AznAttrList object.

attr_names **- output**
AznStrings object for the returned array of attribute names.

Remarks

This method returns the set of all attribute names in the specified attribute list. The *attr_names* object is both an input and output parameter. The returned array of attribute names is set in this object.

Return Values

Status return code which can be passed to the error_major and error_minor methods to retrieve the Azn major and minor error code values.

attrlist_name_get_num

Returns the number of value attributes for a specified name attribute in a specified attribute list.

Syntax

```
public static native int attrlist_name_get_num(AznAttrList attr_list,
                                               String attr_name,
                                               AznInteger num_values)
```

Parameters

attr_list **- input**

AznAttrList object.

attr_name **- input**

Name attribute for the entry whose number of value attributes is to be returned.

num_values **- input /output**

AznInteger object for the number of value attributes returned.

Remarks

This method returns the number of value attributes for a specified name attribute in a specified attribute list. The *num_values* object is both an input and output parameter. The returned number of values is set in this object.

Return Values

Status return code which can be passed to the error_major and error_minor methods to retrieve the Azn major and minor error code values.

creds_combine

Combines two authorization credentials chains and a returns the resulting combined credentials chain.

Syntax

```
public static native int creds_combine(AznCreds creds,
                                       AznCreds creds_to_add,
                                       AznCreds combined_creds)
```

Parameters

creds **- input**

AznCreds object for the credentials chain whose first indexed entry is the credential of the initiator of the request.

creds_to_add **- input**

AznCreds object for the credentials to be added to an existing credentials chain.

combined_creds **- output**

AznCreds object for the new credentials chain, which consists of the credentials chain referenced by *creds* followed by the credentials chain referenced by *creds_to_add*.

Remarks

This method takes a AznCreds object *creds_to_add*, which refers to a credentials chain, and adds it to the end of a chain of one or more credentials, which are referenced by the AznCreds object *creds*. The credentials chain referenced by *creds* must contain as its first indexed credential the credentials of the initiator. The credentials chain referenced by *creds* might also contain the (previously combined) credentials of one or more of the initiator's proxies.

The combined credentials is returned through the AznCreds object *combined_creds*.The *combined_creds* object is both an input and output parameter. The handle to the resulting combined credentials chain is e set in this object.

Return Values

Status return code which can be passed to the error_major and error_minor methods to retrieve the Azn major and minor error code values.

creds_create

Creates a new, empty credentials chain.

Syntax

```
public static native int creds_create(AznCreds creds)
```

Parameters

creds **- input /output**

AznCreds object for the new empty credentials structure that is returned.

Remarks

This method creates a new, empty credentials chain The *creds* object is both an input and output parameter. The handle to the new empty credentials structure is set in this object.

Return Values

Status return code which can be passed to the error_major and error_minor methods to retrieve the Azn major and minor error code values.

creds_delete

Deletes a credentials chain.

Syntax

```
public static native int creds_delete(AznCreds creds)
```

Parameters

creds **- input /output**

AznCreds object.

Remarks

This method deletes a credentials chain. The *creds* object is both an input and output parameter. The handle to credentials structure is in this object to an invalid value to ensure that it cannot be used in future calls.

Return Values

Status return code which can be passed to the error_major and error_minor methods to retrieve the Azn major and minor error code values.

---

creds_for_subject

Obtains a specified credentials chain from a combined credentials chain.

Syntax

```
public static native int creds_for_subject(AznCreds combined_creds,
                                           int subject_index,
                                           AznCreds new_creds)
```

Parameters

combined_creds **- input**

AznCreds object representing a credentials chain which contains one or more individual credentials structures. When this method returns, the structure referred to by c*ombined_creds* is unchanged.

subject_index **- input**

Index of the requested individual credentials chain within the combined credentials chain. The index of the first credentials chain in the combined credentials chain, which should be that of the initiator, is zero (0).

new_creds **- input/output**

AznCreds object for the returned credentials structure.

Remarks

This method sets the object *new_creds* to a credentials chain for the individual credential at index *subject_index* within the credentials chain *combined_creds*. The chain *combined_creds* contains the combined credentials of several subjects.

This method does not modify the *combined_creds* credentials chain.

The *new_creds* object is both an input and output parameter which will be set with the handle to the requested credentials structure.

Combined credentials chains are created by Azn.creds_combine. The first credential chain in a combined credentials chain is that of the initiator, and its index is zero (0).

Use Azn.creds_num_of_subjects to determine the total number of credentials chains in a combined credentials chain.

Return Values

Status return code which can be passed to the error_major and error_minor methods to retrieve the Azn major and minor error code values.

creds_get_attrlist_for_subject

Returns attribute information from a specified subject's credentials chain within a specified (and possibly combined) credentials chain.

Syntax

```
public static native int creds_get_attrlist_for_subject(
                                                AznCreds creds,
                                                int subject_index,
                                                AznAttrList creds_attrlist)
```

Parameters

creds **- input**

AznCreds object representing a credentials chain which contains one or more individual credentials structures.

subject_index **- input**

Index of the requested individual subject within the credentials chain. The index of the first credential in the combined credentials chain, which should be that of the initiator, is zero (0).

creds_attrlist **- input /output**

AznAttrList object for the returned attribute list.

Remarks

This method returns an attribute list containing privilege attribute information from the credentials chain for the individual subject at index *subject_index* within a credentials chain *creds*.

The first credential chain in a combined credentials chain is that of the initiator, and its index will be zero (0).

The *creds_attrlist* object is both an input and output parameter which is set with the handle to an attribute list containing the attribute information from the specified credentials structure.

Use the *Azn.attrlist** methods to retrieve individual attribute values from *creds_attrlist*.

Return Values

Status return code which can be passed to the error_major and error_minor methods to retrieve the Azn major and minor error code values.

creds_get_pac

Creates and returns a privilege attribute certificate (PAC) by invoking a specified PAC service on the supplied credentials chain.

Syntax

```
public static native int creds_get_pac(AznCreds creds,
                                       String pac_svc_id,
                                       AznBuffer pac)
```

Parameters

creds **- input**
AznCreds object for the credentials whose information is used to build the PAC.

pac_svc_id **- input**
Identification (id) of the PAC service that produces the PAC.

pac **- input /output**
AznBuffer object for the returned PAC.

Remarks

Create a privilege attribute certificate (PAC) by invoking a specified PAC service on the supplied credentials.

This method uses the PAC service whose identification is supplied as *pac_svc_id* to build a new PAC. The PAC service uses the information in the supplied credentials chain to build the PAC. Different PAC services might produce PACs with different formats. Some PAC services can cryptographically protect or sign the PACs they produce.

When *pac_svc_id* is NULL, the default PAC service returns an architecture-independent and network-independent encoding of the specified credentials chain. This PAC can be safely transmitted. The receiver of the PAC can use Azn.pac_get_creds to decode the PAC and obtain a valid copy of the original credentials chain.

The *pac* object is both an input and output parameter which will be set to contain the new PAC.

Return Values

Status return code which can be passed to the error_major and error_minor methods to retrieve the Azn major and minor error code values.

creds_modify

Modifies an existing credentials chain and returns an object containing a pointer to the handle to a new credentials chain containing the modifications.

Syntax

```
public static native int creds_modify(AznCreds creds,
                                       String mod_svc_id,
                                       AznAttrList mod_info,
                                       AznCreds new_creds)
```

Parameters

creds **- input**
AznCreds object for the credentials to be modified.

mod_svc_id **- input**
Identification (id) of the credential modification service.

mod_info **- input**
AznAttrList object for the attribute list containing modification service-specific or application-specific data that describes the desired credential modifications.

new_creds **- input /output**
AznCreds object for the modified credentials structure handle.

Remarks

This method uses the specified modification service *mod_svc_id*, and optionally an attribute list *mod_info* which contains modification information provided by the caller, to modify a copy of the supplied credentials chain *creds*. The method returns a pointer to a handle to a new credentials chain *new_creds* containing the requested modifications. The supplied credentials chain is unchanged.

When *mod_svc_id* is NULL, this method modifies an existing credential chain *creds* by adding the attribute list *mod_info* to the credentials chain, and returning the modified credential in *new_creds*.

If the input *creds* handle references a combined credentials chain with more than one element, only the first element will be modified. This is the default behavior when *mod_svc_id* is NULL. In this case, the output chain consists of the modified first element followed by unmodified copies of the remaining elements in the input combined credentials chains. The elements in the output credentials chain are kept in the same order as their counterparts in the input credentials chain.

The *new_creds* object is both an input output parameter which will be set to contain the handle to the new credentials structure.

Return Values

Status return code which can be passed to the error_major and error_minor methods to retrieve the Azn major and minor error code values.

creds_num_of_subjects

Returns the number of individual subjects' credentials chains in a combined credentials chain.

Syntax

```
public static native int creds_num_of_subjects(
                                                AznCreds creds,
                                                AznInteger num_of_subjects)
```

Parameters

creds **- input**

AznCreds object for the credentials chain.

num_of_subjects **- input /output**

AznInteger object which is set with the number of subjects whose credentials appear in the input credentials chain *creds*.

Remarks

This method returns the number of individual subjects, *num_of_subjects*, whose credentials appear in a credentials chain *creds*. The *num_of_subjects* object is both an input and output parameter which is set to contain the number of individual credentials.

Return Values

Status return code which can be passed to the error_major and error_minor methods to retrieve the Azn major and minor error code values.

decision_access_allowed

Makes an access control decision.

Syntax

```
public static native int decision_access_allowed(
                                                  AznCreds creds,
                                                  String protected_resource,
                                                  String operation,
                                                  AznInteger permission)
```

Parameters

creds **- input**

AznCreds object for the initiator's credential chain.

protected_resource **- input**

Name of the target resource of the request.

operation **- input**

Name of the requested operation.

permission **- input /output**

AznInteger object where the decision result is returned. If the returned status value is Azn.S_COMPLETE, the returned permission will be Azn.PERMITTED or Azn.NOT_PERMITTED.

If additional information beyond a boolean result is needed, use Azn.decision_access_allowed_ext.

Remarks

This method decides whether the initiator specified by credentials *creds* is authorized to perform the operation *operation* on the target *protected_resource*. The decision is returned through *permission*.

The *permission* object is both an input and output parameter which is set to contain the decision result of Azn.PERMITTED or Azn.NOT_PERMITTED. Calling application are bound by the decision result only when the returned status value is Azn.S_COMPLETE. When the returned status value is not Azn.S_COMPLETE, the permission object does not contain a valid decision result.

Return Values

Status return code which can be passed to the error_major and error_minor methods to retrieve the Azn major and minor error code values.

decision_access_allowed_ext

Makes an access control decision using application-specific context information; returns information about why the decision was made.

Syntax

```
public static native int decision_access_allowed_ext(
                                                AznCreds creds,
                                                String protected_resource,
                                                String operation,
                                                AznAttrList app_context,
                                                AznInteger permission,
                                                AznAttrList permission_info)
```

Parameters

creds **- input**

AznCreds object for the initiator's credentials chain.

protected_resource **- input**

Name of the target of the request.

operation **- input**

Name of the requested operation.

app_context **- input**

AznAttrList object for an attribute list containing application-specific context access control information. A NULL value indicates there is no context access control information.

permission **- input /output**

AznInteger object that contains the decision result. If the returned status value is Azn.S_COMPLETE, the returned permission will be Azn.PERMITTED or Azn.NOT_PERMITTED.

permission_info **- input /output**

AznAttrList object for an attribute list where implementation specific information about the decision can be returned. A null object indicates that no information about the decision is returned.

The parameter *permission_info* can be used to return implementation-specific qualifiers to Azn.NOT_PERMITTED. The qualifiers can be used to assist the calling application or the initiator in formulating a request which will be authorized. Examples of such qualifiers might include: “not permitted yet,” “requires additional privilege attributes,” or “permissible with restrictions.”

Return Values

This method decides whether the initiator specified by the credentials chain *creds* is authorized to perform the operation *operation* on the target *protected_resource*. Optionally, callers can supply application-specific context access control information using the *app_context* argument. The decision is returned through *permission*.

Optionally, the implementation can return implementation-specific information about the decision through *permission_info*. For example, the information can indicate which rule was responsible for granting or denying access.

The *permission* object is both an input and output parameter which is set to contain the decision result of Azn.PERMITTED or Azn.NOT_PERMITTED. Calling application are bound by the decision result only if the returned status value is

Azn.S_COMPLETE. When the returned status value is not Azn.S_COMPLETE, the permission object does not contain a valid decision result.

The *permission_info* object is both an input and output parameter which is used to return implementation specific attribute names and values indicating the reason why the decision was made.

Return Values

Status return code which can be passed to the error_major and error_minor methods to retrieve the Azn major and minor error code values.

error_major

Obtains the major error code associated with a status code that is returned by one of the methods in this class.

Syntax

```
public static native int error_major(int azn_status)
```

Parameters

azn_status **- input**

Previously returned status code by any of the Azn.* methods.

Remarks

This method obtains the major error code associated with a status code that is returned by one of the methods in this class.

Return Values

Major error code for the specified status code.

error_minor

Returns the implementation-specific minor error code that is associated with a status code that was returned by one of the methods in this class.

Syntax

```
public static native int error_minor(int azn_status)
```

Parameters

azn_status **- input**

An Azn status code.

Remarks

This method returns the implementation-specific minor error code that is associated with a status code that was returned by one of the methods in this class.

Return Values

Minor error code for the specified status code.

error_minor_get_string

Returns an object containing the string value for the implementation-specific minor error code that is associated with a status code that was returned by one of the methods in this class.

Syntax

```
public static native int error_minor(int azn_status,
                                     AznString string)
```

Parameters

azn_status **- input**

An Azn status code.

string **- output**

An AznString object containing the string that describes the condition that triggered the generation of the *azn_status* code.

Remarks

This method returns a string that describes the error corresponding to a previously returned minor error status code.

Return Values

String value of the minor error code for the specified status code.

id_get_creds

Returns an object set to the handle to the credentials chain associated by a specified authorization authority with a specified identity.

Syntax

```
public static native int id_get_creds(String authority,
                                      String mechanism_id,
                                      AznAuthInfo mechanism_info,
                                      AznCreds new_creds)
```

Parameters

authority **- input**

Identification (id) of the authorization authority to be used to build the credential. A null input value selects a default.

mechanism_id **- input**

Authentication mechanism that is used to generate the identity passed through the *mechanism_info* object. A null input value selects a default authentication mechanism.

mechanism_info **- input**

AznAuthInfo object containing initiator access control information, which consists of identity information obtained from an authentication service. The authentication service used to produce this information should be identified using the *mechanism_id* parameter. A null input value denotes the default identity for the selected authentication mechanism from the environment.

new_creds **- input /output**

AznCreds object which is set with the handle to a new, empty credentials chain.

Remarks

This method builds an authorization credentials chain, referenced by the returned handle *new_creds*, for the identity corresponding to the initiator access control information *mechanism_info* produced by an authentication mechanism *mechanism_id*.

Specifying a null value for *authority* causes the default authority to be used. The default authority is Policy Director, which is the only authority supported by this release of the Policy Director Authorization API.

Specifying null values for *mechanism_id* and *mechanism_info* causes the default authentication mechanism and the default identity to be the authentication mechanism used in the Policy Director secure domain.

The *new_creds* object is both an input and output parameter which is set to contain the handle to the credentials structure.

Return Values

Status return code which can be passed to the error_major and error_minor methods to retrieve the Azn major and minor error code values.

initialize

Initializes the authorization service.

Syntax

```
public static native int initialize(AznAttrList init_data,
                                    AznAttrList init_info)
```

Parameters

init_data **- input**

AznAttrList object for the attribute list containing implementation-specific initialization data.

init_info **- input /output**

AznAttrList object for the attribute list used to return implementation specific information about the initialization.

Remarks

This method must be called before calling most other Authorization API methods. The exceptions to this rule are the attribute list methods (Azn.attrlist_*) and the error handling methods (Azn.error_*).

The *init_info* object is both an input and output parameter which is set to contain implementation specific information about the initialization.

Return Values

Status return code which can be passed to the error_major and error_minor methods to retrieve the Azn major and minor error code values.

pac_get_creds

Returns a handle to new credentials chain that is derived from a privilege attribute certificate (PAC) by a specified PAC service.

Syntax

```
public static native int pac_get_creds(AznBuffer pac,
                                       String pac_svc_id,
                                       AznCreds new_creds)
```

Parameters

pac **- input**
AznBuffer object that holds the supplied PAC.

pac_svc_id **- input**
Identification (id) of the PAC service that produces the new credentials chain.

new_creds **- output**
AznCreds object to be set with the handle to the new credentials chain.

Remarks

This method uses the identified PAC service (*pac_svc_id*) to build a new credentials chain using the information in the supplied PAC (*pac*). Some PAC services will cryptographically verify the protection or signature on the received PAC, and will return an error if the PAC cannot be verified.

The *new_creds* object is both an input and output parameter which will be set to with the handle to the new credentials.

This method decodes PACs that are built by Azn.creds_get_pac.

Return Values

Status return code which can be passed to the error_major and error_minor methods to retrieve the Azn major and minor error code values.

set_debug_mode

Sets the debug mode for the native method implementation.

Syntax

```
public static native void setDebugMode(int mode)
```

Remarks

When the debug mode is set to 1, the native methods write debug trace information to standard output.

The default is 0 which disables the native method debug trace.

Parameters

mode **- input**
Debug mode

shutdown

Cleans up internal authorization service state in preparation for shutdown.

Syntax

```
public static native int shutdown()
```

Remarks

Use Azn.shutdown to clean up the Authorization API's memory and other internal implementation state before the application exits. This method shuts down the implementation state created by Azn.initialize.

The only authorization API methods that can be used after calling Azn.shutdown, prior to calling Azn.initialize again, are the attribute list methods (Azn.attrlist_*) and the error handling methods (Azn.error_*), and the memory release methods (Azn.*_delete).

Return Values

Status return code which can be passed to the error_major and error_minor methods to retrieve the Azn major and minor error code values.

util_client_authenticate

Performs authentication from a user name and password.

Syntax

```
public static native int util_client_authenticate(
                                                   String principal_name,
                                                   String password)
```

Parameters

principal_name **- input**

Name of the principal (user) to be authenticated.

password **- input**

The password for the user.

Remarks

Performs a login from a user name and password pair. Starts a background thread to refresh the login context as necessary.

The Authorization API must be initialized before this method is called. Use Azn.initialize to initialize the Authorization API.

Return Values

Status return code which can be passed to the error_major and error_minor methods to retrieve the Azn major and minor error code values.

util_password_authenticate

Performs authentication for a user name and password pair, and returns authentication information when the authentication is successful.

Syntax

```
public static native int util_password_authenticate(
                                                    String principal_name,
                                                    String password,
                                                    AznString mechanism_id,
                                                    AznAuthInfo authinfo)
```

Parameters

principal_name **- input**

Name of the user (principal) used to log in. If LDAP authentication is used, this will be a DN string.

password **- input**

Password for the user.

mechanism_id **- input /output**

AznString object set with the mechanism ID identifying the authentication mechanism.

authinfo **- input /output**

AznAuthInfo object set with the results of the authentication when the authentication is successful.

Remarks

This method performs authentication for a user name and password pair, and returns authentication information when authentication is successful.

The authentication mechanism used depends upon the underlying authentication mechanism that was configured when the Authorization API was installed. Policy Director supports DCE and LDAP authentication. For LDAP Authorization API authentication, the Azn.initialize method must have completed successfully.

This method does not establish a security context for the application.

The *mechanism_id* object is both an input and output parameter that is set with the mechanism ID for the authentication mechanism.

The *authinfo* object is both an input and output parameter that is set with the results of a successful authentication.

The *mechanism_id* and *authinfo* returned can be appended with data specific to the principal and passed into the Azn.id_get_creds method.

Return Values

Status return code which can be passed to the error_major and error_minor methods to retrieve the Azn major and minor error code values.

util_server_authenticate

Performs authentication from a keytab file, and starts a background thread to refresh the login context as necessary.

Syntax

```
public static native int util_server_authenticate(
                                                   String principal_name,
                                                   String keytab_path)
```

Parameters

principal_name **- input**

Name of the user (principal) to be authenticated.

keytab_path **- input**

Path to the keytab file containing the principal's key.

Remarks

This method performs authentication from a keytab file, and starts a background thread to refresh the login context as necessary.

In order to use this utility method, applications that operate in a Policy Director secure domain that uses an LDAP user registry must use DCE commands to create a keytab file.

The Authorization API must be initialized before this method is called. Use Azn.initialize to initialize the Authorization API.

Return Values

Status return code which can be passed to the error_major and error_minor methods to retrieve the Azn major and minor error code values.

Class com.ibm.pd.Authzn.AznAttrList

```
public class AznAttrList extends Object
```

Description

The AznAttrList class implements an attribute list. Attribute lists are represented in the Authorization C APIs by the datatype azn_attrlist_h_t). An object of this class simply contains the handle to an Attribute List and is used as either an input or output parameter for the methods that create, use, modify or delete an attribute list.

Variable Index

Constructor Index

Variable

handle

Syntax

```
public long handle
```

Remarks

Attribute list handle.

Constructors

AznAttrList

Syntax

```
public AznAttrList()
```

Remarks

Constructor for an AznAttrList object. This constructor initializes the Attribute List handle to 0.

AznAttrList(long)

Syntax

```
public AznAttrList(long)
```

Remarks

Constructor for an AznAttrList object. This constructor takes the Attribute List handle as a parameter.

Class com.ibm.pd.Authzn.AznAuthInfo

```
public class AznAuthInfo extends Object
```

Description

The AznAuthInfo class implements the access control information that is passed as input to the Azn.id_get_creds method within the *mechanism_info* parameter.

Objects of this class represent one of the data structures used by the Authorization C APIs for the following data types:

C API Data Type	Usage
azn_authdce_t	For DCE credentials
azn_authldap_t	For LDAP credentials
azn_unauth_t	For unauthenticated credentials

Variable Index

Constructor Index

Variables

user_identity

Syntax

```
public String user_identity
```

Remarks

DCE principal name or LDAP distinguished name. This variable is not used for unauthenticated credentials.

auth_method

Syntax

```
public String auth_method
```

Remarks

A string containing authentication method identification for DCE or LDAP credentials. The content of the string is defined by the application. This variable is not used for unauthenticated credentials.

ipaddr

Syntax

```
public long ipaddr
```

Remarks

IP address of requesting user.

qop

Syntax

```
public String qop
```

Remarks

Quality of protection that is required for requests that are made by this user.

user_info

Syntax

```
public String user_info
```

Remarks

Additional user information that might be required for auditing.

browser_info

Syntax

```
public String browser_info
```

Remarks

Browser (if any) that is employed by the user.

authnmech_info

Syntax

```
public String authnmech_info
```

Remarks

Additional authentication mechanism information. Supplied and used as needed by the application. This variable is not used for unauthenticated credentials.

Constructor

AznAuthInfo

Syntax

```
public AznAuthInfo()
```

Remarks

Constructor for an AznAuthInfo object which initializes all the data members to 0 or null.

Class com.ibm.pd.Authzn.AznBuffer

```
public class AznBuffer extends Object
```

Description

The AznBuffer class implements a binary buffer value. The buffer value is represented in the Authorization C APIs by the data type azn_buffer_t.

An object of this class contains a single data member which is a byte array. The byte array is used as either an input or output parameter for the Azn methods that require a buffer value.

Variable Index

Constructor Index

Variable

value

Syntax

```
public byte value[]
```

Remarks

The byte array containing the buffer value.

Constructor

AznBuffer

Syntax

```
public AznBuffer()
```

Remarks

Constructor for an AznBuffer object which initializes the byte array value to null.

Class com.ibm.pd.Authzn.AznCreds

```
public class AznCreds extends Object
```

Description

The AznCreds class implements an authorization credentials. The authorization credentials is represented in the Authorization C APIs by the data type azn_creds_h_t.

An object of this class simply contains the handle to a credentials structure. An AznCreds object is used as either an input or output parameter for the methods that create or use authorization credentials.

Variable Index

- "handle" on page 72

  Credentials structure handle

Constructor Index

- "AznCreds" on page 72

  Constructor for an AznCreds object which initializes the credentials structure handle to 0.

- "AznCreds(long)" on page 72

  Constructor for an AznCreds object which takes the credentials structure handle as a parameter.

Variable

handle

Syntax

```
public long handle
```

Remarks

Credentials structure handle.

Constructors

AznCreds

Syntax

```
public AznCreds()
```

Remarks

Constructor for an AznCreds object which initializes the credentials structure handle to 0.

AznCreds(long)

Syntax

```
public AznCreds(long value)
```

Remarks

Constructor for an AznCreds object which takes the credentials structure handle as a parameter.

Parameters

handle **- output**

Credentials structure handle.

Class com.ibm.pd.Authzn.AznInteger

```
public class AznInteger extends Object
```

Description

The AznInteger class implements an object used to return an integer value.

An object of this class simply contains the integer value which is an output parameter for the methods that return an integer value.

Variable Index

Constructor Index

Variable

value

Syntax

```
public int value
```

Remarks

Integer value.

Constructor

AznInteger

Syntax

```
public AznInteger()
```

Remarks

Constructor for an AznInteger object which initializes the integer value to 0.

Class com.ibm.pd.Authzn.AznString

```
public class AznString extends Object
```

Description

The AznString class implements an object used to return a string value.

An object of this class simply contains the string value which is an output parameter for the methods that return a string value.

Variable Index

Constructor Index

Variable

value

Syntax

```
public String value
```

Remarks

String value.

Constructor

AznString

Syntax

```
public AznString()
```

Remarks

Constructor for an AznString object which initializes the string value to null.

Class com.ibm.pd.Authzn.AznStrings

```
public class AznStrings extends Object
```

Description

The AznStrings class implements an object used to return an array of string value.

An object of this class simply contains the string array which is an output parameter for the methods that return an array of string values.

Variable Index

Constructor Index

Variable

value

Syntax

```
public String value[]
```

Remarks

Array of string values.

Constructor

AznStrings

Syntax

```
public AznStrings()
```

Remarks

Constructor for an AznString object which initializes the string array to null.

Index

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

Y

Appendix. Notices

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:
INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the information. IBM may make improvements and / or changes in the product(s) and / or the program(s) described in this information at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Department LZKS
11400 Burnet Road
Austin, TX 78758
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurement may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. _*enter the year or years.*_ All rights reserved.

Trademarks

The following terms are trademarks of International Business Machines Corporation in the United States, or other countries, or both:

AIX
FirstSecure
IBM
SecureWay

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark in the United States, other countries, or both and is licensed exclusively through The Open Group.

Other company, product, and service names may be trademarks or service marks of others.

Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.