Tivoli SecureWay Policy Director
Authorization API Java Wrappers Developer Reference

Copyright Notice
©Copyright IBM Corporation 2001 All rights reserved. May only be used pursuant to a Tivoli Systems Software License Agreement, an IBM Software License Agreement, or Addendum for Tivoli Products to IBM Customer or License Agreement. No part of this publication may be reproduced, transmitted, transcribed, stored in a retrieval system, or translated into any computer language, in any form or by any means, electronic, mechanical, magnetic, optical, chemical, manual, or otherwise, without prior written permission of IBM Corporation. IBM Corporation grants you limited permission to make hardcopy or other reproductions of any machine-readable documentation for your own use, provided that each such reproduction shall carry the IBM Corporation copyright notice. No other rights under copyright are granted without prior written permission of IBM Corporation. The document is not intended for production and is furnished “as is” without warranty of any kind. All warranties on this document are hereby disclaimed, including the warranties of merchantability and fitness for a particular purpose.
U.S. Government Users Restricted Rights—Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corporation.

Trademarks
IBM, the IBM logo, Tivoli, the Tivoli logo, AIX, Policy Director, and SecureWay are trademarks or registered trademarks of International Business Machines Corporation or Tivoli Systems Inc. in the United States, other countries, or both.
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 of The Open Group in the United States and other countries.
Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.
Other company, product, and service names may be trademarks or service marks of others.

Notices
References in this publication to Tivoli Systems or IBM products, programs, or services do not imply that they will be available in all countries in which Tivoli Systems or IBM operates. Any reference to these products, programs, or services is not intended to imply that only Tivoli Systems or IBM products, programs, or services can be used. Subject to valid intellectual property or other legally protectable right of Tivoli Systems or IBM, any functionally equivalent product, program, or service can be used instead of the referenced product, program, or service. The evaluation and verification of operation in conjunction with other products, except those expressly designated by Tivoli Systems or IBM, are the responsibility of the user. Tivoli Systems or 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 the IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, New York 10504-1785, U.S.A.
Contents

Preface ................................................................................................. i
Who Should Read This Guide............................................................. i
What This Guide Contains .................................................................. i
Typeface Conventions ......................................................................... ii
Related Policy Director Documents .................................................... iii
Accessing Online Documentation ...................................................... iv
Ordering Documentation ....................................................................... iv
Providing Feedback about Product Documentation ............................. v
Contacting Customer Support .......................................................... v

Chapter 1 — Authorization API

1.1 Introducing the Authorization API.................................................... 1-2
  1.1.1 The Open Group Authorization API standard............................. 1-3
  1.1.2 Policy Director Authorization API Version History ..................... 1-5
  1.1.3 The Policy Director Authorization Model ................................. 1-5

1.2 Installing the Java Authorization API .............................................. 1-6

1.3 Building Applications with the Authorization API ......................... 1-6
  1.3.1 Software Requirements ....................................................... 1-6
  1.3.2 Setting Environment Variables .............................................. 1-7

1.4 Introducing the Java Authorization API Classes and Methods ......... 1-8
  1.4.1 Class com.ibm.pd.Authzn.Azn............................................... 1-9
  1.4.2 Class com.ibm.pd.Authzn.AznString ....................................... 1-12
  1.4.3 Class com.ibm.pd.Authzn.AznStrings ..................................... 1-12
  1.4.4 Class com.ibm.pd.Authzn.AznInteger ..................................... 1-12
  1.4.5 Class com.ibm.pd.Authzn.AznBuffer ..................................... 1-12
  1.4.6 Class com.ibm.pd.Authzn.AznAttrList ................................... 1-13
  1.4.7 Class com.ibm.pd.Authzn.AznCreds ....................................... 1-14
  1.4.8 Class com.ibm.pd.Authzn.AznAuthInfo ................................... 1-15
  1.4.9 Status Codes and Error Handling ......................................... 1-16

1.5 Summarizing Authorization API Tasks ......................................... 1-17
  1.5.1 Required Tasks ..................................................................... 1-17
  1.5.2 Optional Tasks ...................................................................... 1-17
  1.5.3 Runtime Environment ......................................................... 1-17
1.6 Authenticating an API Application

1.6.1 Using srvsslcfg

1.6.2 Using bassslcfg

1.7 Initializing the Authorization Service

1.7.1 Specifying an Authorization API Configuration File

1.7.2 Specifying the Maximum Number of Handle Groups

1.7.3 Specifying the Type of Cache Mode

1.7.4 Configuring Local Cache File Names

1.7.5 Configuring Local Cache Refresh

1.7.6 Configuring Local Cache Notification Listener

1.7.7 Configuring Notification Listener Ports

1.7.8 Configuring SSL from the API Client to Policy Director

1.7.9 Configuring the Authorization API for LDAP Access

1.7.10 Configuring LDAP Access over SSL

1.7.11 Configuring Advanced LDAP Parameters

1.7.12 Specifying LDAP User Registry Replica Access

1.7.13 Enabling the Return of Permission Information

1.7.14 Starting the Authorization Service

1.8 Verifying the Identity of a User

1.9 Obtaining User Authorization Credentials

1.9.1 Specifying the Authorization Authority

1.9.2 Specifying Authentication User Registry Type

1.9.3 Specifying User Authentication Identity

1.9.4 Specifying Additional User Information

1.9.5 Obtaining Authorization Credentials for the User

1.10 Obtaining an Authorization Decision

1.10.1 Mapping the User Operation to a Policy Director Permission

1.10.2 Mapping the Requested Resource to a Protected Object

1.10.3 Assigning the User Credentials to a Credentials Handle

1.10.4 Building an Attribute List for Additional Application Information

1.10.5 Obtaining an Authorization Decision

1.11 Cleaning Up and Shutting Down

1.11.1 Releasing Allocated Memory

1.11.2 Shutting Down the Authorization API
1.12 Handling Credentials ................................................................................... 1-56
  1.12.1 Converting Credentials to a Transportable Format ........................ 1-56
  1.12.2 Converting Credentials to the Native Format ............................. 1-57
  1.12.3 Creating a Chain of Credentials .............................................. 1-57
  1.12.4 Determining the Number of Credentials in a Credentials Chain.... 1-58
  1.12.5 Obtaining a Credential from a Chain of Credentials............... 1-58
  1.12.6 Modifying the Contents of a Credential ................................... 1-59
  1.12.7 Obtaining an Attribute List from a Credential ....................... 1-60
  1.12.8 Comparing Two Credentials ................................................. 1-62
  1.12.9 Copying A Credential............................................................. 1-62

1.13 Deploying Applications with the Authorization API .................................. 1-63
  1.13.1 Software Requirements .............................................................. 1-63
  1.13.2 Example program authzn_demo ............................................... 1-63

Chapter 2 — Java Wrappers Reference

cred_version ................................................................................................. 2-12
cred_mech_id .............................................................................................. 2-12
cred_principal_uuid ..................................................................................... 2-12
cred_principal_name .................................................................................. 2-12
cred_group_uuids ....................................................................................... 2-12
cred_group_names ....................................................................................... 2-12
cred_dce_name .......................................................................................... 2-13
cred_ldap_dn .............................................................................................. 2-13
cred_urof_name ......................................................................................... 2-13
cred_user_info .......................................................................................... 2-13
cred_auth_method .................................................................................... 2-13
cred_authnmech_info ................................................................................ 2-14
cred_ip_address ......................................................................................... 2-14
cred_qop_info .......................................................................................... 2-14
cred_browser_info ..................................................................................... 2-14
perminfo_all_attrs ................................................................................... 2-14
perminfo_qop ............................................................................................ 2-15
perminfo_qop_uint ............................................................................... 2-15
perminfo_qop_uint_none ................................................................. 2-15
perminfo_qop_uint_integrity ............................................................. 2-15
perminfo_qop_uint_privacy .............................................................. 2-15
perminfo_wm .......................................................................................... 2-15
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>perminfo_wm_permitted</td>
<td>2-16</td>
</tr>
<tr>
<td>perminfo_al</td>
<td>2-16</td>
</tr>
<tr>
<td>perminfo_al_none</td>
<td>2-16</td>
</tr>
<tr>
<td>perminfo_al_permit</td>
<td>2-16</td>
</tr>
<tr>
<td>perminfo_al_deny</td>
<td>2-16</td>
</tr>
<tr>
<td>perminfo_al_error</td>
<td>2-16</td>
</tr>
<tr>
<td>perminfo_al_admin</td>
<td>2-16</td>
</tr>
<tr>
<td>perminfo_al_all</td>
<td>2-16</td>
</tr>
<tr>
<td>svc_version</td>
<td>2-17</td>
</tr>
<tr>
<td>pac_svc_dce_epac</td>
<td>2-17</td>
</tr>
<tr>
<td>ent_svc_pd_pobj</td>
<td>2-18</td>
</tr>
<tr>
<td>ent_svc_pd_pobj_path</td>
<td>2-18</td>
</tr>
<tr>
<td>ent_svc_pd_pobj_reqd_ops</td>
<td>2-18</td>
</tr>
<tr>
<td>ent_svc_pd_pobj_matches</td>
<td>2-18</td>
</tr>
<tr>
<td>PERMITTED</td>
<td>2-19</td>
</tr>
<tr>
<td>NOT_PERMITTED</td>
<td>2-19</td>
</tr>
<tr>
<td>IV_UNAUTH</td>
<td>2-19</td>
</tr>
<tr>
<td>IV_DCE</td>
<td>2-19</td>
</tr>
<tr>
<td>IV_LDAP</td>
<td>2-19</td>
</tr>
<tr>
<td>IV_URAF</td>
<td>2-20</td>
</tr>
<tr>
<td>operation_attach</td>
<td>2-20</td>
</tr>
<tr>
<td>operation_bypasstod</td>
<td>2-20</td>
</tr>
<tr>
<td>operation_browse</td>
<td>2-20</td>
</tr>
<tr>
<td>operation_control</td>
<td>2-21</td>
</tr>
<tr>
<td>operation_traverse</td>
<td>2-21</td>
</tr>
<tr>
<td>operation_delegation</td>
<td>2-21</td>
</tr>
<tr>
<td>operation_view</td>
<td>2-21</td>
</tr>
<tr>
<td>operation_modify</td>
<td>2-21</td>
</tr>
<tr>
<td>operation_delete</td>
<td>2-21</td>
</tr>
<tr>
<td>operation_server_admin</td>
<td>2-22</td>
</tr>
<tr>
<td>operation_audit</td>
<td>2-22</td>
</tr>
<tr>
<td>operation_integrity</td>
<td>2-22</td>
</tr>
<tr>
<td>operation_list_directory</td>
<td>2-22</td>
</tr>
<tr>
<td>operation_password</td>
<td>2-22</td>
</tr>
<tr>
<td>operation_create</td>
<td>2-22</td>
</tr>
<tr>
<td>operation_add</td>
<td>2-23</td>
</tr>
<tr>
<td>operation_privacy</td>
<td>2-23</td>
</tr>
<tr>
<td>operation_read</td>
<td>2-23</td>
</tr>
<tr>
<td>operation_execute</td>
<td>2-23</td>
</tr>
<tr>
<td>operation_list_directory</td>
<td>2-23</td>
</tr>
<tr>
<td>operation_connect</td>
<td>2-23</td>
</tr>
<tr>
<td>operation_forward</td>
<td>2-24</td>
</tr>
<tr>
<td>Function</td>
<td>Page</td>
</tr>
<tr>
<td>-----------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>init_mode</td>
<td>2-24</td>
</tr>
<tr>
<td>init_qop</td>
<td>2-24</td>
</tr>
<tr>
<td>init_cfg_file</td>
<td>2-24</td>
</tr>
<tr>
<td>init_max_handle_groups</td>
<td>2-25</td>
</tr>
<tr>
<td>init_set_perminfo_attrs</td>
<td>2-25</td>
</tr>
<tr>
<td>init_db_file</td>
<td>2-25</td>
</tr>
<tr>
<td>init_audit_file</td>
<td>2-26</td>
</tr>
<tr>
<td>init_cache_refresh_interval</td>
<td>2-26</td>
</tr>
<tr>
<td>init_listen_flags</td>
<td>2-26</td>
</tr>
<tr>
<td>init_namespace_location</td>
<td>2-26</td>
</tr>
<tr>
<td>init_tcp_port</td>
<td>2-27</td>
</tr>
<tr>
<td>init_udp_port</td>
<td>2-27</td>
</tr>
<tr>
<td>init_ldap_bind_dn</td>
<td>2-27</td>
</tr>
<tr>
<td>init_ldap_bind_pwd</td>
<td>2-27</td>
</tr>
<tr>
<td>init_ldap_ssl_keyfile</td>
<td>2-27</td>
</tr>
<tr>
<td>init_ldap_host</td>
<td>2-28</td>
</tr>
<tr>
<td>init_ldap_port</td>
<td>2-28</td>
</tr>
<tr>
<td>init_ldap_ssl_keyfile</td>
<td>2-28</td>
</tr>
<tr>
<td>init_ldap_ssl_keyfile_dn</td>
<td>2-28</td>
</tr>
<tr>
<td>init_ldap_ssl_keyfile_pwd</td>
<td>2-28</td>
</tr>
<tr>
<td>init_ldap_replica</td>
<td>2-29</td>
</tr>
<tr>
<td>init_ldap_cache</td>
<td>2-29</td>
</tr>
<tr>
<td>init_ldap_prefer_rw_svr</td>
<td>2-29</td>
</tr>
<tr>
<td>init_ldap_max_search_size</td>
<td>2-29</td>
</tr>
<tr>
<td>init_ldap_use_compare</td>
<td>2-30</td>
</tr>
<tr>
<td>init_ent_svc</td>
<td>2-30</td>
</tr>
<tr>
<td>init_pac_svc</td>
<td>2-30</td>
</tr>
<tr>
<td>init_cred_mod_svc</td>
<td>2-30</td>
</tr>
<tr>
<td>init_remote_ns_loc</td>
<td>2-30</td>
</tr>
<tr>
<td>Azn</td>
<td>2-31</td>
</tr>
<tr>
<td>attrlist_add_entry</td>
<td>2-31</td>
</tr>
<tr>
<td>attrlist_add_entry_buffer</td>
<td>2-32</td>
</tr>
<tr>
<td>attrlist_create</td>
<td>2-33</td>
</tr>
<tr>
<td>attrlist_delete</td>
<td>2-33</td>
</tr>
<tr>
<td>attrlist_delete_entry</td>
<td>2-34</td>
</tr>
<tr>
<td>attrlist_copy</td>
<td>2-34</td>
</tr>
<tr>
<td>attrlist_get_entry_buffer_value</td>
<td>2-35</td>
</tr>
<tr>
<td>attrlist_get_entry_string_value</td>
<td>2-36</td>
</tr>
<tr>
<td>attrlist_get_names</td>
<td>2-37</td>
</tr>
<tr>
<td>attrlist_name_get_num</td>
<td>2-38</td>
</tr>
<tr>
<td>creds_combine</td>
<td>2-39</td>
</tr>
<tr>
<td>creds_copy</td>
<td>2-40</td>
</tr>
<tr>
<td>Class</td>
<td>Page</td>
</tr>
<tr>
<td>-------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>handle</td>
<td>2-67</td>
</tr>
<tr>
<td>AznCreds</td>
<td>2-68</td>
</tr>
<tr>
<td>AznCreds(long)</td>
<td>2-68</td>
</tr>
<tr>
<td>Class com.ibm.pd.Authzn.AznInteger</td>
<td>2-69</td>
</tr>
<tr>
<td>value</td>
<td>2-69</td>
</tr>
<tr>
<td>AznInteger</td>
<td>2-69</td>
</tr>
<tr>
<td>Class com.ibm.pd.Authzn.AznString</td>
<td>2-70</td>
</tr>
<tr>
<td>value</td>
<td>2-70</td>
</tr>
<tr>
<td>AznString</td>
<td>2-70</td>
</tr>
<tr>
<td>Class com.ibm.pd.Authzn.AznStrings</td>
<td>2-71</td>
</tr>
<tr>
<td>value</td>
<td>2-71</td>
</tr>
<tr>
<td>AznStrings</td>
<td>2-71</td>
</tr>
</tbody>
</table>
Preface

This document contains information about a Java implementation of the Policy Director Authorization API. This Java implementation consists of Java classes and methods that function as “wrappers” around the C API functions. Java programmers can use these Java wrappers to add Policy Director authorization to Java applications.

Who Should Read This Guide

The target audience for this installation guide includes:

- Security administrators
- System installation and deployment administrators
- Network system administrators
- IT architects
- Application developers

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

What This Guide Contains

This document contains the following chapters:

- Chapter 1, "Authorization API"
  This chapter describes the Authorization API model, and describes the Policy Director implementation of Java wrappers around the Policy Director Authorization C API.

- Chapter 2, "Java Wrappers Reference"
  This chapter provides reference pages for each of the classes, constructors, methods, and variables in the Java wrappers implementation.
Typeface Conventions

This guide uses several typeface conventions for special terms and actions. These conventions have the following meaning:

**Bold** Command names and options, keywords, and other information that you must use literally appear in **bold**.

*Italics* Variables, command arguments, and values you must provide appear in *italics*. Titles of publications and special words or phrases that are emphasized also appear in *italics*.

**Monospace** Code examples, command lines, screen output, and system messages appear in monospace font.
Related Policy Director Documents

The following table summarizes the available Policy Director documentation:

<table>
<thead>
<tr>
<th>Tivoli SecureWay Policy Director Technical Documents</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Installation Guides</strong></td>
</tr>
<tr>
<td>Tivoli SecureWay Policy Director Base for AIX Installation Guide</td>
</tr>
<tr>
<td>Tivoli SecureWay Policy Director Base for HP-UX Installation Guide</td>
</tr>
<tr>
<td>Tivoli SecureWay Policy Director Base for Solaris Installation Guide</td>
</tr>
<tr>
<td>Tivoli SecureWay Policy Director Base for Windows Installation Guide</td>
</tr>
<tr>
<td>Tivoli SecureWay Policy Director WebSEAL Installation Guide</td>
</tr>
<tr>
<td>Tivoli SecureWay Policy Director NetSEAL Installation Guide</td>
</tr>
<tr>
<td>Tivoli SecureWay Policy Director Management Console Installation Guide</td>
</tr>
<tr>
<td><strong>Administration Guides</strong></td>
</tr>
<tr>
<td>Tivoli SecureWay Policy Director Base Administration Guide</td>
</tr>
<tr>
<td>Tivoli SecureWay Policy Director WebSEAL Administration Guide</td>
</tr>
<tr>
<td>Tivoli SecureWay Policy Director NetSEAL Administration Guide</td>
</tr>
<tr>
<td>Tivoli SecureWay Policy Director Management Console Administration Guide</td>
</tr>
<tr>
<td><strong>Developer References</strong></td>
</tr>
<tr>
<td>Tivoli SecureWay Policy Director Authorization ADK Developer Reference</td>
</tr>
<tr>
<td>Tivoli SecureWay Policy Director WebSEAL Developer Reference</td>
</tr>
<tr>
<td><strong>Supplemental Documentation</strong> (updated regularly on the Tivoli support site)</td>
</tr>
<tr>
<td>Tivoli SecureWay Policy Director Release Notes</td>
</tr>
<tr>
<td>Tivoli SecureWay Policy Director Lotus Domino Registry Supplement</td>
</tr>
<tr>
<td>Tivoli SecureWay Policy Director Performance Tuning Guide</td>
</tr>
</tbody>
</table>
Accessing Online Documentation

The Tivoli Customer Support Web site ([http://www.tivoli.com/support/](http://www.tivoli.com/support/)) provides links to the following documentation information:

- Technical information, including release notes, installation and configuration guides, administration guides, and developer references.
- Frequently Asked Questions (FAQs)
- Software download information


You can access the index of online Tivoli publications at [http://www.tivoli.com/support/documents/](http://www.tivoli.com/support/documents/). Click on Master Index to find product-specific support pages.


The documentation for some products is available in PDF and HTML formats. Translated documents are also available for some products.

To access most of the documentation, you need an ID and a password. To obtain an ID for use on the support Web site, go to [http://www.tivoli.com/support/getting/](http://www.tivoli.com/support/getting/).

Resellers should refer to [http://www.tivoli.com/support/smb/index.html](http://www.tivoli.com/support/smb/index.html) for more information about obtaining Tivoli technical documentation and support.

Business Partners should refer to the Preface section entitled “Ordering Documentation” for more information about obtaining Tivoli technical documentation.

Ordering Documentation

Order Tivoli documentation online at [http://www.tivoli.com/support/Prodman/html/pub_order.html](http://www.tivoli.com/support/Prodman/html/pub_order.html) or by calling one of the following telephone numbers:

- U.S. customers: (800) 879-2755
- Canadian customers: (800) 426-4968
Providing Feedback about Product Documentation

We are very interested in hearing about your experience with Tivoli products and documentation, and we welcome your suggestions for improvements. If you have comments or suggestions about our products and documentation, contact us in one of the following ways:

- Send e-mail to pubs@tivoli.com.
- Fill out our customer feedback survey at http://www.tivoli.com/support/survey/.

Contacting Customer Support

The Tivoli Customer Support Handbook at:
http://www.tivoli.com/support/handbook/

provides information about all aspects of Tivoli Customer Support, including the following:

- Registration and eligibility
- How to contact support, depending on the severity of your problem
- Telephone numbers and e-mail addresses, depending on the country you are in
- What information you should gather before contacting support
This chapter contains the following topics:

- 1.1 Introducing the Authorization API
- 1.2 Installing the Java Authorization API
- 1.3 Building Applications with the Authorization API
- 1.4 Introducing the Java Authorization API Classes and Methods
- 1.5 Summarizing Authorization API Tasks
- 1.6 Authenticating an API Application
- 1.7 Initializing the Authorization Service
- 1.8 Verifying the Identity of a User
- 1.9 Obtaining User Authorization Credentials
- 1.10 Obtaining an Authorization Decision
- 1.11 Cleaning Up and Shutting Down
- 1.12 Handling Credentials
- 1.13 Deploying Applications with the Authorization API
1.1 Introducing the Authorization API

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).
1.1.1 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.

![Diagram of the ISO 10181-3 Authorization Model]

Figure 1-1: The ISO 10181-3 Authorization Model
Chapter 1: Authorization API

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

![Figure 1-2: The Policy Director Implementation of the ISO Authorization Model](image)

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.
1.1.2 Policy Director Authorization API Version History


The C implementation of the Policy Director Authorization API is described in the Policy Director Programming Guide and Reference, Version 3.7. 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.

1.1.3 The Policy Director Authorization Model

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 1 in the Policy Director Base 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 1 in the Policy Director Base Administration Guide.

3. Develop your application logic to enforce the security policy.
1.2 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 Tivoli Web site:

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 Tivoli Web site.

1.3 Building Applications with the Authorization API

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

- **Software Requirements (Section 1.3.1)**
- **Setting Environment Variables (Section 1.3.2)**

Note: Instructions in this section refer to non-DCE environments. To develop DCE applications, use the Policy Director Version 3.6 java wrappers.

1.3.1 Software Requirements

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 Base components installed:

- Policy Director Runtime
- Policy Director Management Server
- Policy Director Authorization Server
- Policy Director Application Development Kit
You should also install the Policy Director management utility:
- Policy Director Management Console

When 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 installation guide for the Policy Director Base distribution for your platform.

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 Runtime
- Policy Director Application Development Kit

### 1.3.2 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.
   - On HP-UX systems only, add to SHLIB_PATH the name of the directory containing the file libaznjni.sl.
1.4 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 Authorization ADK Developer Reference.

Note: Policy Director also provides a separate Java implementation of the Authorization API, based on the J2EE security model. This implementation is recommended for adding authorization capability to new Java application code. For more information, see the Policy Director Authorization ADK Developer 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 (Section 1.4.1)
- Class com.ibm.pd.Authzn.AznString (Section 1.4.2)
- Class com.ibm.pd.Authzn.AznStrings (Section 1.4.3)
- Class com.ibm.pd.Authzn.AznInteger (Section 1.4.4)
- Class com.ibm.pd.Authzn.AznBuffer (Section 1.4.5)
- Class com.ibm.pd.Authzn.AznAttrList (Section 1.4.6)
- Class com.ibm.pd.Authzn.AznCreds (Section 1.4.7)
- Class com.ibm.pd.Authzn.AznAuthInfo (Section 1.4.8)
1.4.1 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 Authorization ADK Developer 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 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

<table>
<thead>
<tr>
<th>Method</th>
<th>Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>attrlist_add_entry</td>
<td>Class com.ibm.pd.Authzn.AznAttrList (Section 1.4.6)</td>
</tr>
<tr>
<td>attrlist_add_entry_buffer</td>
<td></td>
</tr>
<tr>
<td>attrlist_copy</td>
<td></td>
</tr>
<tr>
<td>attrlist_create</td>
<td></td>
</tr>
<tr>
<td>attrlist_delete</td>
<td></td>
</tr>
<tr>
<td>attrlist_delete_entry</td>
<td></td>
</tr>
<tr>
<td>attrlist_get_entry_buffer_value</td>
<td></td>
</tr>
<tr>
<td>attrlist_get_entry_string_value</td>
<td></td>
</tr>
<tr>
<td>attrlist_get_names</td>
<td></td>
</tr>
<tr>
<td>attrlist_name_get_num</td>
<td></td>
</tr>
<tr>
<td>util_handle_is_valid</td>
<td></td>
</tr>
</tbody>
</table>
### Credentials

<table>
<thead>
<tr>
<th>method</th>
<th>Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>creds_combine</td>
<td>Creating a Chain of Credentials (Section 1.12.3)</td>
</tr>
<tr>
<td>creds_compare</td>
<td>Comparing Two Credentials (Section 1.12.8)</td>
</tr>
<tr>
<td>creds_copy</td>
<td>Copying A Credential (Section 1.12.9)</td>
</tr>
<tr>
<td>creds_create</td>
<td>Obtaining User Authorization Credentials (Section 1.9)</td>
</tr>
<tr>
<td>creds_delete</td>
<td>Releasing Allocated Memory (Section 1.11.1)</td>
</tr>
<tr>
<td>creds_for_subject</td>
<td>Obtaining a Credential from a Chain of Credentials (Section 1.12.5)</td>
</tr>
<tr>
<td>creds_get_attrlist_for_subject</td>
<td>Obtaining an Attribute List from a Credential (Section 1.12.7)</td>
</tr>
<tr>
<td>creds_get_pac</td>
<td>Converting Credentials to a Transportable Format (Section 1.12.1)</td>
</tr>
<tr>
<td>creds_modify</td>
<td>Modifying the Contents of a Credential (Section 1.12.6)</td>
</tr>
<tr>
<td>creds_num_of_subject</td>
<td>Determining the Number of Credentials in a Credentials Chain (Section 1.12.4)</td>
</tr>
<tr>
<td>id_get_creds</td>
<td>Obtaining User Authorization Credentials (Section 1.9)</td>
</tr>
<tr>
<td>pac_get_creds</td>
<td>Converting Credentials to the Native Format (Section 1.12.2)</td>
</tr>
<tr>
<td>util_handle_is_valid</td>
<td>Class com.ibm.pd.Authzn.AznCreds (Section 1.4.7)</td>
</tr>
</tbody>
</table>
Introducing the Java Authorization API Classes and Methods

Authorization Decisions

<table>
<thead>
<tr>
<th>method</th>
<th>Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>decision_access_allowed</td>
<td>Obtaining an Authorization Decision (Section 1.10)</td>
</tr>
<tr>
<td>decision_access_allowed_ext</td>
<td></td>
</tr>
</tbody>
</table>

Initialization, Shutdown, and Error Handling

<table>
<thead>
<tr>
<th>Method</th>
<th>Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>error_major</td>
<td>Status Codes and Error Handling (Section 1.4.9)</td>
</tr>
<tr>
<td>error_minor</td>
<td></td>
</tr>
<tr>
<td>error_minor_get_string</td>
<td></td>
</tr>
<tr>
<td>initialize</td>
<td>Initializing the Authorization Service (Section 1.7)</td>
</tr>
<tr>
<td>shutdown</td>
<td>Shutting Down the Authorization API (Section 1.11.2)</td>
</tr>
<tr>
<td>util_errcode</td>
<td>Status Codes and Error Handling (Section 1.4.9)</td>
</tr>
</tbody>
</table>

API Extensions

<table>
<thead>
<tr>
<th>Method</th>
<th>Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>util_errcode</td>
<td>Status Codes and Error Handling (Section 1.4.9)</td>
</tr>
<tr>
<td>util_handle_is_valid</td>
<td>Class com.ibm.pd.Authzn.AznCreds (Section 1.4.7)</td>
</tr>
<tr>
<td>util_password_authenticate</td>
<td>Verifying the Identity of a User (Section 1.8)</td>
</tr>
</tbody>
</table>
Chapter 1: Authorization API

1.4.2 **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:

```java
AznString testuser = new AznString();
testuser.value = 'user_name';
```

1.4.3 **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.

1.4.4 **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.

1.4.5 **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.

Azn.Buffer objects are used output parameters to the Azn.attrlist_get_entry_buffer_value and Azn.creds_get_pac methods.
1.4.6 **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:

<table>
<thead>
<tr>
<th>Task</th>
<th>Description</th>
</tr>
</thead>
</table>
| Create an attribute list      | Use `attrlist_create` 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 `attrlist_add_entry` to add a string name-value pair.  
  Use `attrlist_add_entry_buffer` to add a buffer name-value pair (AznBuffer object). |
| Delete an entry from an attribute list | Use `attrlist_delete_entry` to delete all the values that are assigned to an attribute in an attribute list. |
Chapter 1: Authorization API

<table>
<thead>
<tr>
<th>Task</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get attribute names from an attribute list</td>
<td>Use <code>attrlist_get_names</code> to get all the names in an attribute list. The names are returned as an array of strings in an AznStrings object.</td>
</tr>
<tr>
<td>Get the number of values for a specified attribute name</td>
<td>Use <code>attrlist_name_get_num</code> to get the number, as an integer, of the value attributes for a specified name in the attribute list.</td>
</tr>
<tr>
<td>Copy an attribute list.</td>
<td>Use <code>attrlist_copy</code> to copy one attribute list to a new attribute list.</td>
</tr>
<tr>
<td>Get a value</td>
<td>Use <code>attrlist_get_entry_string_value</code> to get the value attribute of a string of type <code>azn_string_t</code> for a specified name in an attribute list.</td>
</tr>
<tr>
<td></td>
<td>Use <code>attrlist_get_entry_buffer_value</code> to get the value attribute of a buffer of type <code>azn_buffer_t</code> 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.</td>
</tr>
<tr>
<td>Delete an attribute list</td>
<td>Use <code>attrlist_delete</code> to delete the attribute list associated with a specified attribute list handle.</td>
</tr>
<tr>
<td>Determine if the attribute list handle is valid</td>
<td>Use <code>util_handle_is_valid</code> to determine if an attribute list handle is associated with valid data.</td>
</tr>
</tbody>
</table>

1.4.7 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 `azncreds_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 `creds_create` 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 `creds_delete` on page 53 to release the memory allocated for the credential structure.

To determine if a credentials handle is valid, use the Authorization API utility method `util_handle_is_valid`.

### 1.4.8 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:

<table>
<thead>
<tr>
<th>C API Data Type</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>azn_authdce_t</td>
<td>For DCE credentials</td>
</tr>
<tr>
<td>azn_authldap_t</td>
<td>For LDAP credentials</td>
</tr>
<tr>
<td>azn_unauth_t</td>
<td>For unauthenticated credentials</td>
</tr>
</tbody>
</table>
1.4.9 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 `error_major` to extract major error codes from the returned status. Major error codes are defined according to the The Open Group Authorization API Standard.

Use `error_minor` 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 `error_minor_get_string` to obtain string values for the minor error codes returned by Azn.error_minor.

Use `util_errcode` to build an `azn_status_t` error code from a major and minor status. Use this to return standardized error codes to Authorization API applications when developing Authorization Service Plug-ins.

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 API Java Wrappers distribution.
1.5 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.

1.5.1 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:

- Authenticating an API Application (Section 1.6)
- Initializing the Authorization Service (Section 1.7)
- Verifying the Identity of a User (Section 1.8)
- Obtaining User Authorization Credentials (Section 1.9)
- Obtaining an Authorization Decision (Section 1.10)
- Cleaning Up and Shutting Down (Section 1.11)

1.5.2 Optional Tasks

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

- Handling Credentials (Section 1.12)

1.5.3 Runtime Environment

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

- Deploying Applications with the Authorization API (Section 1.13)
1.6 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.

Use the svrsslcfg utility to accomplish the above tasks. Run this utility before initializing the Authorization API.

1.6.1 Using svrsslcfg

Use the svrsslcfg utility to create a user identity for the application, and to configure the SSL communication between the application and the Policy Director Management Server.

The svrsslcfg utility performs the following tasks:

- Creates a user identity for the application based on the specified server. For example, demo_user.

- Creates an SSL key file for that user: For example, demo_user.key and demo_user.sth.

- Adds the user ivacld-servers group for a server type of local, or to the remote-acl-users group for a server type of remote.
The syntax for `svrsslcfg` is:

```
svrsslcfg <cfg_file> -config -d kdb_dir -n server_name -s server_type -P admin_pwd [-A admin_id] \ 
[-r port] [-t timeout] [-e pwd_life] [-C cert_file]
```

The following example is used by the Authorization Demo program that is distributed as part of the Policy Director Authorization ADK:

```
svrsslcfg aznAPI.conf -config -d /opt/PolicyDirector/authzn_adk/example -n authzn_demo \ 
-S <svr-password> -s local -P <sec_master password>
```

The above example specifies the following configuration information:

- The configuration file is `aznAPI.conf`
- Configure (`-config`) the application to communicate with the Policy Director Management Server
- The directory for the keyfile is: `/opt/PolicyDirector/authzn_adk/example`
- The name of the application server is `authzn_demo`.
- The application server password is `<svr-password>`.
- The server type is `local`, rather than `remote`.
- The LDAP administrator password is `<sec_master password>`.

For more information, see the reference page for `svrsslcfg` in the *Policy Director Authorization ADK Developer Reference*.

### 1.6.2 Using bassslcfg

Use the `bassslcfg` utility to enable the Authorization API application to communicate over SSL with a Policy Director Authorization server that is located on another machine in the secure domain.

For more information, see the reference page for `bassslcfg` in the *Policy Director Authorization ADK Developer Reference*.
Chapter 1: Authorization API

1.7 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.

There are two ways to specify the initialization data:
- Specify input arguments to `Azn.initialize`.
- Specify entries in the Authorization API configuration file.

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`.

Each input argument to `Azn.initialize` has a corresponding entry in the Authorization API configuration file. Input arguments take precedence over configuration file entries.

You can define entries in the configuration file, and then use input parameters to `Azn.initialize` to override them as needed when each Authorization API application initializes.

In order to use a configuration file, specify the configuration file location as an input parameter to `Azn.initialize`.

Complete the instructions in the following sections:
- Specifying an Authorization API Configuration File (Section 1.7.1)
- Specifying the Maximum Number of Handle Groups (Section 1.7.2)
- Specifying the Type of Cache Mode (Section 1.7.3)
- Configuring Local Cache File Names (Section 1.7.4)
- Configuring Local Cache Refresh (Section 1.7.5)
- Configuring Local Cache Notification Listener (Section 1.7.6)
- Configuring Notification Listener Ports (Section 1.7.7)
- Configuring SSL from the API Client to Policy Director (Section 1.7.8)
- Configuring the Authorization API for LDAP Access (Section 1.7.9)
- Configuring LDAP Access over SSL (Section 1.7.10)
- Configuring Advanced LDAP Parameters (Section 1.7.11)
- Specifying LDAP User Registry Replica Access (Section 1.7.12)
- Enabling the Return of Permission Information (Section 1.7.13)
- Starting the Authorization Service (Section 1.7.14)
1.7.1 Specifying an Authorization API Configuration File

You can specify a configuration file that contains initialization values. The configuration file is a text file consisting of stanzas. Each stanza contains a series of name = value pairs. Each of the pairs corresponds to an input parameter that can get passed to `Azn.initialize`.

If no configuration file is specified, `Azn.initialize` obtains initialization parameters only from the attribute list contained in the `init_data` input parameter.

There is no configuration file specified by default.

To specify the location of a configuration file:

1. Call `attrlist_create` to create a new attribute list called `init_data`. This method returns an AznAttrList object.

2. Use `attrlist_add_entry` to add the attribute `azn_init_cfg_file` and assign it a value:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Azn.init_cfg_file</code></td>
<td><code>&lt;filename&gt;</code></td>
<td>A configuration file containing initialization values for the Policy Director Authorization API. There is no default value. A sample configuration file name could be: <code>aznapi.conf</code></td>
</tr>
</tbody>
</table>
1.7.2 Specifying the Maximum Number of Handle Groups

You can specify the maximum number of handle groups to allocate, to tune your application for memory usage or performance. Each handle group can contain up to 4096 handles. Handle groups are used to manage the allocation of handles for credentials and attribute lists.

<table>
<thead>
<tr>
<th>Entry</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attribute List Entry:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Azn.init_max_handle_groups</td>
<td>&lt;integer&gt;</td>
<td>Integer value of 0 (zero) means to use the default value. The default number of handle groups is 256. This enables the use of 1,000,000 handles. Minimum number of groups is 1. Maximum number of groups is 1,048,574. Use of the default value is recommended.</td>
</tr>
<tr>
<td>Configuration File Entry:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>max-handle-groups</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Using An Attribute List

Use `attrlist_add_entry` to add the attribute `Azn.init_max_handle_groups` and assign it a valid value.

Using A Configuration File

Go to the `[aznapi-configuration]` stanza in the configuration file and assign the `max-handle-groups` entry a valid value.
1.7.3 **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.

- When using remote mode, the caller of the Authorization API must be a member of the `remote-acl-users` group.
  
  For more information on remote cache mode, see Chapter 1 of the *Policy Director Base Administration Guide*.

- When using local mode, the caller of the Authorization API must be a member of the `ivacld-servers` group.
  
  For more information on local cache mode, see Chapter 1 of the *Policy Director Base Administration Guide*.

The `svrsslcfg` utility creates a user identity (or server principal) for the caller and automatically adds it to the appropriate group. The `svrsslcfg` utility determines which group membership is required, based on whether you specified `local` or `remote` to the `-s` parameter. For more information, see the `svrsslcfg` reference page in the *Policy Director Authorization ADK Developer Reference*.

You can specify the cache mode either as an input parameter to `Azn.initialize()` or as a configuration file entry. The following table displays the names of the cache mode entries and the valid values.

<table>
<thead>
<tr>
<th>Entry</th>
<th>Valid Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attribute List Entry: Azn.init_mode</td>
<td>local</td>
<td>The Policy Director Authorization Service runs in the same server process as the application using the Authorization API.</td>
</tr>
<tr>
<td>Configuration File Entry: mode</td>
<td>remote</td>
<td>The Policy Director Authorization Service runs as a different server process from the application using the Authorization API.</td>
</tr>
</tbody>
</table>
Using An Attribute List

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

1. Call `attrlist_create` to create a new attribute list called `init_data`. This method returns a pointer to an attribute list handle.

   **Note:** Skip this step if the list was created in Section 1.7.1: “Specifying an Authorization API Configuration File”.

2. Use `attrlist_add_entry` to add the attribute `Azn.init_mode` and assign it a valid value.

Using A Configuration File

Go to the `[aznapi-configuration]` stanza in the configuration file and assign a valid value to the `mode` entry.
1.7.4 Configuring Local Cache File Names

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 either by adding attributes to the init_data attribute list or by setting entries in the Authorization API configuration file.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attribute List Entry: Azn.init_db_file</td>
<td>filename</td>
<td>Path name to the persistent authorization policy database replica.</td>
</tr>
<tr>
<td>Configuration File Entry: db-file</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Attribute List Entry: Azn.init_audit_file</td>
<td>filename</td>
<td>Path and file name for the file that collects Authorization API audit events.</td>
</tr>
<tr>
<td>Configuration File Entry: auditlog</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Using An Attribute List

Use Azn.attrlist_add_entry() to specify path names for files used by the authorization service.

Using A Configuration File

Go to the [aznapi-configuration] stanza in the configuration file and specify path names for the db-file and auditlog files used by the authorization service.
1.7.5 Configuring Local Cache Refresh

<table>
<thead>
<tr>
<th>Attribute List Entry:</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Azn.init_cache_refresh_interval</td>
<td>disable</td>
<td>Refreshing of the local authorization policy database disabled.</td>
</tr>
<tr>
<td>default</td>
<td>600 seconds.</td>
<td></td>
</tr>
<tr>
<td>number of seconds</td>
<td>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.</td>
<td></td>
</tr>
</tbody>
</table>

Using An Attribute List

Use `Azn.attrlist_add_entry()` to enable the Authorization API to poll the master authorization database.

Using A Configuration File

Go to the `[aznapi-configuration]` stanza in the configuration file and specify valid values for the `cache-refresh-interval` entry. Note that values can be either `disable`, `default`, or the number of seconds expressed as an integer.
1.7.6 Configuring Local Cache Notification Listener

<table>
<thead>
<tr>
<th>Attribute List Entry: Azn.init_listen_flags</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configuration File Entry: listen-flags</td>
<td>enable</td>
<td>Enable the notification listener.</td>
</tr>
</tbody>
</table>

Using An Attribute List

Use `Azn.attrlist_add_entry()` to configure the notification listener.

Using A Configuration File

Go to the `[aznapi-configuration]` stanza in the configuration file and specify valid values for the `listen-flags` entry. You can enter a combination of values. For example:

```
listen-flags = enable dynamic_port_selection use_tcp_port
```
1.7.7 Configuring Notification Listener Ports

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

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configuration File Entry:</td>
<td>port number</td>
<td>Use this value to specify the TCP port on which the application will listen for notifications from the master database that is has changed. All communications on this port are SSL encrypted.</td>
</tr>
<tr>
<td>ssl-listening-port</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Using A Configuration File

Go to the [ssl] stanza in the configuration file and specify a port number for the ssl-listening-port. The value should be non-zero and not used by any other service on the computer.
1.7.8 Configuring SSL from the API Client to Policy Director

You can specify a number of attributes or configuration file entries that describe the SSL communications configuration between the Authorization API Client, running in remote mode, and the Policy Director Authorization Server and Policy Director Management Server.

The following configuration file entries in the following table are found in the [ssl] stanza.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attribute List Entry: Azn.init_ssl_keyfile</td>
<td>&lt;keyfile-path&gt;</td>
<td>This is the keyfile used to communicate with ivmgrd/ivacl. It will be created by the svrsslcfg utility.</td>
</tr>
<tr>
<td>Configuration File Entry: ssl-keyfile</td>
<td>Any relative or fully qualified filename.</td>
<td></td>
</tr>
<tr>
<td>Attribute List Entry: Azn.init_ssl_stashfile</td>
<td>&lt;keyfile-path&gt;</td>
<td>This the stash file for the keyfile. It is created by the svrsslcfg utility. It is used as an obfuscated password to the keyfile. This file should be appropriately secured.</td>
</tr>
<tr>
<td>Configuration File Entry: ssl-keyfile-stash</td>
<td>Any relative or fully qualified filename.</td>
<td></td>
</tr>
<tr>
<td>Configuration File Entry: ssl-keyfile-label</td>
<td>Any string</td>
<td>The label of the certificate to use within the keyfile. In normal operation this is not used. However it is useful if the keyfiles are constructed outside of the svrsslcfg utility and contain multiple certificates.</td>
</tr>
</tbody>
</table>
### Attribute List Entry: `Azn.init_ssl_timeout`
#### Configuration File Entry: `ssl-v3-timeout`
- **Value:** Any non-negative integer. Default value is 7200 seconds
- **Description:** This is the amount of time before an SSL session will expire. The Policy Director Authorization API client automatically creates a new SSL session with new keys when a session expires. This value only applies to the listening aspect of the authorization API’s (when the Policy Director Management Server is calling the application). When the application is calling the Policy Director Management Server or the Policy Director Authorization Server, the session timeout value is dictated by that server.

### Attribute List Entry: `Azn.init_ssl_pwd_life`
#### Configuration File Entry: `ssl-pwd-life`
- **Value:** Any non-negative integer. Default value is 186 days.
- **Description:** This is the amount of time before the password or stash file to the keyfile will expire. The Policy Director Authorization API client automatically refreshes the password or stash file before this expiry time, provided it is in operation.
**Initializing the Authorization Service**

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Configuration File Entry: authn-type</strong></td>
<td>The type of authentication. Possible values are: certificate, password, none. The default value is none.</td>
<td>The method that the Policy Director Management Server will use to authenticate the authorization API client. If the value is certificate, the Policy Director Management Server will map the certificate provided by the authorization API client into an identity and authenticate against it. Note that even if password or none are specified, the client will still need a certificate to communicate with the server. Furthermore, there are currently no operations that can be performed by the API successfully with an authentication type of none.</td>
</tr>
<tr>
<td>Configuration File Entry: authn-user</td>
<td>Any string.</td>
<td>The user name and password that are used if the authentication type is password. It may be unwise to store these in the configuration file, however they can be useful for testing communications.</td>
</tr>
</tbody>
</table>
Chapter 1: Authorization API

Using An Attribute List

Use `Azn.attrlist_add_entry()` to add attributes that contain configuration information for the SSL connections between the API client and the Policy Director servers.

Using A Configuration File

1. Go to the `[ssl]` stanza in the configuration file.
2. Add entries as specified in the preceding table.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attribute List Entry:</td>
<td>Azn.init_ssl_mgr_config</td>
<td>This entry is used to point to the configuration file that was created as part of configuring the Policy Director Runtime Environment (pd.conf). If it is specified, the values for master-host, master-port, and master-dn will come from the manager stanza of pd.conf and override any values specified in the authorization API client's configuration file. Furthermore, if entries or values are not found in pd.conf for any of these entries, empty values will be used.</td>
</tr>
<tr>
<td>Configuration File Entry:</td>
<td>ssl-mgr-config</td>
<td>The relative or fully qualified pathname to the pd.conf file. The pd.conf usually lives in the Policy Director installation directory, under ./lib/pd.conf</td>
</tr>
</tbody>
</table>
Communications Attributes for the Management Server

The configuration file entries described in the following table are found in the [manager] stanza.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attribute List Entry: Azn.init_master_host</td>
<td>&lt;Management Server hostname&gt;</td>
<td>Specifies the hostname of the Policy Director Management Server. This entry and stanza can be in either the authorization API client's configuration file (aznAPI.conf) or the Policy Director Runtime Environment configuration file (pd.conf). The pd.conf value overrides that in aznAPI.conf.</td>
</tr>
<tr>
<td>Configuration File Entry: master-host</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Attribute List Entry: Azn.init_master_port</td>
<td>&lt;port number&gt;</td>
<td>Specifies the port of the Policy Director Management Server. This entry and stanza can be in either the authorization API client's configuration file (aznAPI.conf) or the Policy Director Runtime Environment configuration file (pd.conf). The pd.conf value overrides that in aznAPI.conf.</td>
</tr>
<tr>
<td>Configuration File Entry: master-port</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Using A Configuration File

1. Go to the `[manager]` stanza in the configuration file.

2. Assign values for the Management Server, as specified in the preceding table.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Attribute List Entry:</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Azn.init_master_dn</td>
<td><code>&lt;distinguished name&gt;</code></td>
<td>Specifies the Distinguished Name (DN) of the Policy Director Management Server. The authorization API client can match this value against that provided by the Policy Director Management Server at runtime to prevent spoofing. If the value is empty, no checking will be performed. This entry and stanza can be in either the authorization API client's configuration file (aznAPI.conf) or the Policy Director Runtime Environment configuration file (pd.conf). The pd.conf value overrides that in aznAPI.conf.</td>
</tr>
<tr>
<td><strong>Configuration File Entry:</strong></td>
<td>master-dn</td>
<td></td>
</tr>
</tbody>
</table>
Specifying an Authorization Server Replica

The following configuration file entry is found in the [manager] stanza:

<table>
<thead>
<tr>
<th>Configuration File Entry</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value: replica</td>
<td>Each of these entries specifies a Policy Director Authorization Server. These are usually added to pd.conf via the bassslcfg command. They can be manually added, and there can be more than one entry. Note that the separator for the fields is a colon (&quot;:&quot;) and not a comma (&quot;,&quot;) like the LDAP replicas use.</td>
</tr>
<tr>
<td>&lt;Authorization Server values&gt;</td>
<td>The replicas are of the format: &lt;replica hostname&gt;:&lt;port&gt;:&lt;preference&gt;:&lt;replica cert dn&gt; where: • &lt;replica hostname&gt; is the network name of the server. • &lt;port&gt; is the port on the server. • &lt;preference&gt; is a ranking for attempting contact from 1 to 10 where 10 is the highest preference. • &lt;DN&gt; is the DN for the ivacld server. For example, &quot;rweber.bball.com:7137:5:cn=ivacld/rweber,o=Policy Director,C=US&quot;</td>
</tr>
</tbody>
</table>

Using A Configuration File

1. Go to the [manager] stanza in the configuration file.
2. Assign values for the Authorization Server, using the format specified in the preceding table.
1.7.9 Configuring the Authorization API 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.

<table>
<thead>
<tr>
<th>Entry</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configuration File Entry: enable</td>
<td>yes</td>
<td>Enable LDAP user registry support. This is the default value. This entry is not used when building an attribute list. LDAP access is automatically enabled when the attribute Azn.init_ldap_port is not null.</td>
</tr>
<tr>
<td>Attribute List Entry: Azn.init_ldap_host</td>
<td>host name</td>
<td>Host name of LDAP server.</td>
</tr>
<tr>
<td>Configuration File Entry: host</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Attribute List Entry: Azn.init_ldap_port</td>
<td>port number</td>
<td>Port number for communicating with the LDAP server.</td>
</tr>
<tr>
<td>Configuration File Entry: port</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Attribute List Entry: Azn.init_ldap_bind_dn</td>
<td>LDAP DN</td>
<td>Distinguished Name of the LDAP user. Created by the svrsslcfg utility.</td>
</tr>
<tr>
<td>Configuration File Entry: bind-dn</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Attribute List Entry: Azn.init_ldap_bind_pwd</td>
<td>password</td>
<td>Password for the LDAP user. Created by the svrsslcfg utility.</td>
</tr>
<tr>
<td>Configuration File Entry: bind-pwd</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Using An Attribute List

Use `Azn.attrlist_add_entry()` to add the attributes to the `init_data` attribute list.

Using A Configuration File

1. Go to the `[ldap]` stanza in the configuration file.
2. Add entries for the LDAP server, as described in the preceding table.
1.7.10 Configuring LDAP Access over SSL

If the communication between the Policy Director Authorization server and the LDAP server is over Secure Sockets Layer (SSL), set the following values:

<table>
<thead>
<tr>
<th>Entry</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configuration File Entry:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ssl-enabled</td>
<td>yes</td>
<td>Enables SSL communication with the LDAP server. This entry is not used when building attribute lists. If Azn.init_ldap_ssl_keyfile is not null, then SSL is automatically configured.</td>
</tr>
<tr>
<td>Configuration File Entry:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ssl-keyfile</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Attribute List Entry:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Azn.init_ldap_ssl_keyfile</td>
<td>filename</td>
<td>Name of the SSL key file.</td>
</tr>
<tr>
<td>Configuration File Entry:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ssl-keyfile-dn</td>
<td>KeyLabel</td>
<td>Key label to identify the client certificate that is presented to the LDAP server.</td>
</tr>
<tr>
<td>Attribute List Entry:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Azn.init_ldap_ssl_keyfile_pwd</td>
<td>password</td>
<td>Password to access the SSL key file.</td>
</tr>
<tr>
<td>Configuration File Entry:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ssl-keyfile-pwd</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Note that the Policy Director Authorization API client must use two key files: one for communicating with the LDAP server and one for the Policy Director servers.

Using An Attribute List

Use `Azn.attrlist_add_entry()` to add the attributes to the `init_data` attribute list.

Using a Configuration File

1. Go to the `[ldap]` stanza in the configuration file.
2. Add the SSL values as described in the preceding table.
### 1.7.11 Configuring Advanced LDAP Parameters

<table>
<thead>
<tr>
<th>Entry</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attribute List Entry: Azn.init_ldap_max_search_size</td>
<td></td>
<td>Optional. Limit for the maximum search buffer size returned from the LDAP server in entries. Note that this value can also be limited by the LDAP server itself.</td>
</tr>
<tr>
<td>Configuration File Entry: max-search-size</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Attribute List Entry: Azn.init_ldap_cache</td>
<td>true</td>
<td>Optional. Enable LDAP client-side caching of user, group and LDAP policy data to improve performance for similar LDAP queries.</td>
</tr>
<tr>
<td>Configuration File Entry: cache-enabled</td>
<td>false</td>
<td>Optional. Disable LDAP client-side caching. This is the default value.</td>
</tr>
<tr>
<td>Attribute List Entry: Azn.init_prefer_rw_svr</td>
<td>true</td>
<td>Optional. The client attempts to query the read/write LDAP server (see ldap-replica configuration option) before querying any read-only servers that are configured in the domain.</td>
</tr>
<tr>
<td>Configuration File Entry: prefer-readwrite-server</td>
<td>false</td>
<td>Optional. Do not query read/write LDAP server first</td>
</tr>
<tr>
<td>Attribute List Entry: Azn.init_auth_using_compare</td>
<td>true</td>
<td>Optional. Choose whether ldap_compare() is used instead of ldap_bind() to authenticate LDAP users. This option changes the method used by the Policy Director Authorization API call and Azn.util_password_authenticate().</td>
</tr>
<tr>
<td>Configuration File Entry: auth-using-compare</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>false</td>
<td>Optional. Use ldap_bind().</td>
</tr>
</tbody>
</table>

**Using An Attribute List**

Use `Azn.atrlist_add_entry()` to add advanced LDAP parameters to the attribute list.

**Using a Configuration File**

Go to the [ldap] stanza in the configuration file and specify the advanced LDAP parameters as described in the table above.
1.7.12 Specifying LDAP User Registry Replica Access

Using An Attribute List

Use `Azn.attrlist_add_entry()` to add the attributes that define the LDAP user registry replicas in the domain. Add the attributes to the `init_data` attribute list.

Using a Configuration File

1. Go to the `[ldap]` stanza in the configuration file.
2. Define the LDAP user registry replicas in the domain by specifying valid values for `ldap-replica`.
   
   Assign multiple values to `ldap-replica` by entering a list consisting of entries that are separated by commas. For example:

   ```plaintext
   ldap-replica = barney,391,readwrite,2
   ldap-replica = fred,391,readonly,3
   ```

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attribute List Entry:</td>
<td><code>&lt;ldap-server&gt;</code></td>
<td>The network name of the LDAP server.</td>
</tr>
<tr>
<td>Azn.init ldap replica</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Configuration File Entry:</td>
<td><code>&lt;port&gt;</code></td>
<td>The port on the LDAP server.</td>
</tr>
<tr>
<td>ldap-replica</td>
<td><code>&lt;type&gt;</code></td>
<td>Either “readonly” or “readwrite”</td>
</tr>
<tr>
<td></td>
<td><code>&lt;pref&gt;</code></td>
<td>A preference or priority level to assign to accessing this replica. The minimum value is 1. The maximum value is 10. A higher number denotes a higher preference.</td>
</tr>
</tbody>
</table>
1.7.13 Enabling the Return of Permission Information

You can specify information to be returned by the `Azn.decision_access_allowed_ext` method. This call returns a `AznAttrList` object named `permission_info`.

This attribute list is empty by default but can optionally be used to return additional information about the access control decision. The Authorization API defines a number of permission information attributes that can be returned. In order for these attributes to be returned, they must be set when the Authorization API is initialized.

**Note:** You can also specify that user-defined attributes be returned in `permission_info`. See the `Azn.decision_access_allowed_ext` reference page for more information.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attribute List Entry:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Azn.init_set_perminfo_attrs</td>
<td>Azn.perminfo_all_attrs</td>
<td>Include all of the permission information attributes</td>
</tr>
<tr>
<td>Configuration File Entry:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>permission-info-returned</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Azn.perminfo_wm</td>
<td>Warning mode. when warning mode is enabled, access is always granted. If access should not have been granted then the access is logged.</td>
</tr>
<tr>
<td></td>
<td>Azn.perminfo_wm_permitted</td>
<td>Access permitted because of warning mode. The boolean indicator is used to tell the caller that access was granted because warning mode is enabled.</td>
</tr>
<tr>
<td></td>
<td>Azn.perminfo_al</td>
<td>Auditing events that are performed for this authorization check.</td>
</tr>
</tbody>
</table>
Using An Attribute List Entry as Initialization Data

Use `Azn.attrlist_add_entry()` to add the `Azn.init_set_perminfo_attrs` attributes to the `init_data` attribute list.

Using A Configuration File

1. Go to the `[aznapi-configuration] stanza in the configuration file.

2. Set values for the `permission-info-returned` entry.
1.7.14 Starting the Authorization Service

Complete the following steps:

1. Ensure that the attribute list `init_data` 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(init_data, init_info);
   if (status != Azn.S_COMPLETE)
       return(status);
   ```

   In the example code above, `Azn.initialize()` returns the attribute list `init_info`. 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()`.

When using this method on Windows, do not call it from dllMain(). For more information, see the reference page for `initialize`. 
1.8 Verifying the Identity of a User

The application must verify 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 the LDAP User registry. The user is authenticated, and information about the user can be obtained. This information includes, for example, the LDAP Distinguished Name.

- **Unauthenticated**
  In this case, the user’s identity in the secure domain is not specifically registered in the LDAP 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 (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 the reference page for `util_password_authenticate`. After the application has obtained identity information for the user, you can use the Authorization API to obtain authorization credentials for the user.
1.9 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 LDAP user registry 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 (Section 1.12.2).

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

1. Specifying the Authorization Authority (Section 1.9.1)
2. Specifying Authentication User Registry Type (Section 1.9.2)
3. Specifying User Authentication Identity (Section 1.9.3)
4. Specifying Additional User Information (Section 1.9.4)
5. Obtaining Authorization Credentials for the User (Section 1.9.5)
Chapter 1: Authorization API

1.9.1 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.

1.9.2 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 Section 1.8: “Verifying the Identity of a User”.

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:

<table>
<thead>
<tr>
<th>User Registry Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDAP User Registry</td>
<td>Azn.IV_LDAP</td>
</tr>
<tr>
<td>Unauthenticated</td>
<td>Azn.IV_UNAUTH</td>
</tr>
</tbody>
</table>

1.9.3 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.

<table>
<thead>
<tr>
<th>User Identity Type</th>
<th>Variable</th>
<th>String</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDAP User Registry</td>
<td><code>AznAuthInfo.user_identity</code></td>
<td><code>ldap_dn</code></td>
<td><code>cn=root</code></td>
</tr>
<tr>
<td>Unauthenticated User</td>
<td><code>none</code></td>
<td><code>none</code></td>
<td><code>none</code></td>
</tr>
</tbody>
</table>
If the user is unauthenticated, you do not have to load an identity into `AznAuthInfo.user_identity`.

### 1.9.4 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 in as variables in an `AznAuthInfo` object, as described in the table below:

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>auth_method</code></td>
<td>Indicates that the user was authenticated through the LDAP user registry. This value can be any string that is useful to the application. Not used for unauthenticated users.</td>
</tr>
<tr>
<td><code>authnmech_info</code></td>
<td>Additional authentication information. This value can be any string that is useful to the application. Not used for unauthenticated users.</td>
</tr>
<tr>
<td><code>user_info</code></td>
<td>Additional user information for auditing purposes. This string can contain any information that is useful to the application.</td>
</tr>
<tr>
<td><code>browser_info</code></td>
<td>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.</td>
</tr>
<tr>
<td><code>ipaddr</code></td>
<td>The IP address of the user. This is optional information for use by the application.</td>
</tr>
</tbody>
</table>

The `AznAuthInfo` object that contains all of the above user information will be passed as an input parameter to `Azn.id_get_creds`. 
1.9.5 Obtaining Authorization Credentials for the User

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>authority</td>
<td>The authorization authority, as described in Section 1.9.1: “Specifying the Authorization Authority”.</td>
</tr>
<tr>
<td>mechanism_id</td>
<td>The authentication mechanism, as described in Section 1.9.2: “Specifying Authentication User Registry Type”.</td>
</tr>
</tbody>
</table>
| mechanism_info     | An AznAuthInfo object containing user information, as described in the following sections:  

- Specifying User Authentication Identity (Section 1.9.3)  
- Specifying Additional User Information (Section 1.9.4) |

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:

```java
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 = Azn.IV_LDAP;
/* Specify LDAP user name */
ldap_minfo.user_identity = "cn=testuser";
/* Set LDAP user information. These values are placeholders */
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 a null authority*/
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 the reference page for `id_get_creds`. Refer also to the Authorization API demonstration program. See Example program `authzn_demo` (Section 1.13.2).

The application is now ready to submit the authorization request. See Obtaining an Authorization Decision (Section 1.10).
1.10 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 (Section 1.10.1)
- Mapping the Requested Resource to a Protected Object (Section 1.10.2)
- Assigning the User Credentials to a Credentials Handle (Section 1.10.3)
- Building an Attribute List for Additional Application Information (Section 1.10.4)
- Obtaining an Authorization Decision (Section 1.10.5)

1.10.1 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 the reference section for `Class com.ibm.pd.Authzn.Azn`.

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()`.
1.10.2 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.

Complete the following step:

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

1.10.3 Assigning the User Credentials to a Credentials Handle

The authorization credentials for a user obtained in Section 1.9: “Obtaining User Authorization Credentials” 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 Section 1.12.2: “Converting Credentials to the Native Format”.
1.10.4 Building an Attribute List for Additional Application Information

The Policy Director Authorization API provides the extended method \texttt{Azn.decision\_access\_allowed\_ext} for obtaining an access decision. This method extends \texttt{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 \texttt{Azn.attrlist\_create} to create a new, empty attribute list.
2. Use \texttt{Azn.attrlist\_add\_entry} or \texttt{Azn.attrlist\_add\_entry\_buffer} to add attributes.
3. When all attributes have been added, assign the input parameter \texttt{app\_context} to point to the attribute list.

For more information, see the reference page for \texttt{decision\_access\_allowed\_ext}.
1.10.5 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:

```java
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. 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 output parameter:

<table>
<thead>
<tr>
<th>Type</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Azn.integer</code></td>
<td><code>permission</code></td>
<td>The result of the access request. Consists of one of the following constants:</td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>Azn.PERMITTED</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>Azn.NOT_PERMITTED</code></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Type</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AznAttrList</td>
<td><em>permission_info</em></td>
<td>Application-specific context information contained in attribute list.</td>
</tr>
</tbody>
</table>

For more information, see the reference pages for the following methods:

- `decision_access_allowed`
- `decision_access_allowed_ext`
1.11 Cleaning Up and Shutting Down

The Authorization API provides methods to perform the following clean up and shut down tasks:

- Releasing Allocated Memory (Section 1.11.1)
- Shutting Down the Authorization API (Section 1.11.2)

1.11.1 Releasing Allocated Memory

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

- attrlist_delete
  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.

- attrlist_delete_entry
  Use this method to delete an entry from an attribute list (AznAttrList object). This releases memory allocated by the corresponding Authorization C API.

- creds_delete
  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.

1.11.2 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 to disconnect from and shut down the Authorization API.
1.12 Handling Credentials

In addition to the credentials handling tasks described earlier in this chapter, the Authorization API provides methods to accomplish the following optional tasks:

- Converting Credentials to a Transportable Format (Section 1.12.1)
- Converting Credentials to the Native Format (Section 1.12.2)
- Creating a Chain of Credentials (Section 1.12.3)
- Determining the Number of Credentials in a Credentials Chain (Section 1.12.4)
- Obtaining a Credential from a Chain of Credentials (Section 1.12.5)
- Modifying the Contents of a Credential (Section 1.12.6)
- Obtaining an Attribute List from a Credential (Section 1.12.7)

1.12.1 Converting Credentials to a Transportable Format

Use the method `creds_get_pac` 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 credentials 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.
1.12.2 Converting Credentials to the Native Format

Use the method `pac_get_creds` 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 string `pac_svc_id` to null.
2. Set the input 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.

1.12.3 Creating a Chain of Credentials

Use the method `creds_combine` 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. Assign the AznCreds object `creds_to_prepend` to point to the credentials of the initiator of the request.
2. Assign the AznCreds object `creds_to_add` to point to 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 `combined_creds`. 
1.12.4 Determining the Number of Credentials in a Credentials Chain

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

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

1.12.5 Obtaining a Credential from a Chain of Credentials

Use the method `creds_for_subject` to extract individual credentials from a credentials chain. Credentials chains are created by the `Azn.creds_combine` method.

Complete the following steps:

1. Assign the AznCreds object `creds` with the credentials chain.
2. Assign the AznInteger object `subject_index` 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), pass the number 0 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`.
1.12.6 Modifying the Contents of a Credential

Use the method `creds_modify` to modify a credential by placing additional information, 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` with the new attribute list.

2. Set the credential modification service `mod_svc_id` to null.

3. Set an `AznCreds` object named `creds` to point to the credentials to be modified.

4. Call `Azn.creds_create` to create a new, empty `AznCreds` object.

5. Call `Azn.creds_modify`.

The modified credentials are placed in the `AznCreds` object `newcreds`. 
1.12.7 Obtaining an Attribute List from a Credential

Use the method `creds_get_attrlist_for_subject` 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.

The Authorization API defines a number of attributes that can be returned in the attribute list. Some of the attributes might not be present in an attribute list according to the type of authenticated user and the information that was used when the credential was built.

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Azn.cred_version</td>
<td>The credential version</td>
</tr>
<tr>
<td>Azn.cred_mech_id</td>
<td>The mechanism ID for this credential</td>
</tr>
<tr>
<td>Azn.cred_principal_uuid</td>
<td>The UUID for the entity</td>
</tr>
<tr>
<td>Azn.cred_principal_name</td>
<td>The string name of the entity</td>
</tr>
<tr>
<td>Azn.cred_group_uuids</td>
<td>The string group UUID memberships of this entity</td>
</tr>
<tr>
<td>Azn.cred_group_names</td>
<td>The string group name memberships of this entity</td>
</tr>
<tr>
<td>Azn.cred_ldap_dn</td>
<td>The LDAP DN used to build these authorization credentials</td>
</tr>
<tr>
<td>Azn.cred_uraf_name</td>
<td>The URAF name used to build these authorization credentials</td>
</tr>
<tr>
<td>Azn.cred_user_info</td>
<td>Any user information that was passed in the <code>mechinfo</code> structure</td>
</tr>
<tr>
<td>Azn.cred_auth_method</td>
<td>Any authentication method information that was passed in the <code>mechinfo</code> structure.</td>
</tr>
<tr>
<td>Azn.cred_authmech_info</td>
<td>Any authentication mechanism information that was passed in the <code>mechinfo</code> structure.</td>
</tr>
<tr>
<td>Azn.cred_ip_address</td>
<td>The IP address information passed in the <code>mechinfo</code> structure.</td>
</tr>
<tr>
<td>Azn.cred_browser_info</td>
<td>The browser information passed in the <code>mechinfo</code> structure.</td>
</tr>
</tbody>
</table>
Complete the following steps:

1. Set an AznCreds object *creds* with 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.
1.12.8 Comparing Two Credentials

Use the method `creds_compare` to compare the contents of two credentials.

1. Identify the AznCreds object for each credential to be compared.
2. Pass the AznCreds objects handles as input parameters to `Azn.creds_compare`.
   
   The method `Azn.creds_compare` returns `true` if the credentials are identical or `false` if the credentials differ.

1.12.9 Copying A Credential

Use the method `creds_copy` to copy the contents of one credential to another, new credential.

1. Identify the AznCreds object to be copied.
2. Pass the AznCreds object as the input parameter to `Azn.creds_copy`.
   
   The method `Azn.creds_copy` returns an AznCreds object set with the new credential.
   
   When the application is finished using the new credential, use `Azn.creds_delete` to release the memory that was allocated.

**Note:** To add a credential to an existing credential, use `Azn.creds_combine`. 
1.13 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 (Section 1.13.1)
- Example program authzn_demo (Section 1.13.2)

1.13.1 Software Requirements

1. Add a Java runtime environment in order to run Java programs.

2. Applications that have been developed with the Policy Director Authorization API must be run on systems that are configured into an Policy Director secure domain. When the Policy Director secure domain uses an LDAP user registry, the application deployment system must have an LDAP client installed.

   The minimum Policy Director installation required on a system that will run a Java application is the Policy Director Runtime Environment package.

1.13.2 Example program authzn_demo

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.
Java Wrappers Reference

This 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
- Class com.ibm.pd.Authzn.AznAttrList
- Class com.ibm.pd.Authzn.AznAuthInfo
- Class com.ibm.pd.Authzn.AznCreds
- Class com.ibm.pd.Authzn.AznInteger
- Class com.ibm.pd.Authzn.AznString
- Class com.ibm.pd.Authzn.AznStrings
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

- **cred_auth_method**
  Any authn method info passed in the mech info struct
- **cred_authnmech_info**
  Any authn mechanism info passed in the mech info struct
- **cred_browser_info**
  The browser information passed in the mech info struct
- **cred_dce_name**
  The DCE name used to build these authzn credentials
- **cred_group_names**
  The string group name memberships of this entity
- **cred_group_uuids**
  The string group UUID memberships of this entity
- **cred_ip_address**
  The IP address information passed in the mech info struct
- **cred_ldap_dn**
  The LDAP DN used to build these authzn credentials
- `cred_mech_id`
  The mechanism ID for this cred

- `cred_principal_name`
  The string name of the entity

- `cred_principal_uuid`
  The UUID of the entity

- `cred_qop_info`
  The Quality of Protection information passed in the mech info struct. One of “none”, “integrity” or “privacy”.

- `cred_ufaf_name`
  The URAF name used to build these authzn credentials

- `cred_user_info`
  Any user info passed in the mech info struct

- `cred_version`
  The credential version

- `ent_svc_pd_pobj`
  Entitlements Service ID: Policy Director Protected Object Entitlements.

- `ent_svc_pd_pobj_matches`
  Output attribute: the list of objects which match the input criteria.

- `ent_svc_pd_pobj_path`
  Input attribute: the protected object directory path to be searched.

- `ent_svc_pd_pobj_reqd_ops`
  Input attribute: the requested set of operations for the credential.

- `init_audit_file`
  Attribute name for: Audit path and filename to collect Authorization API audit events.

- `init_cache_refresh_interval`
  Attribute name for: Interval in seconds for local policy cache polled updates.

- `init_cfg_file`
  Attribute name for: Fully qualified Configuration file name.

- `init_cred_mod_svc`
  Credential modification service definition attribute.
Chapter 2: Java Wrappers Reference

- **init_db_file**
  Attribute name for: Path and filename to be used to contain cached authorization policy.

- **init_ent_svc**
  Entitlements service definition attribute.

- **init_ldap_bind_dn**
  Attribute name for: The LDAP distinguished name used to bind to the LDAP server.

- **init_ldap_bind_pwd**
  Attribute name for: The LDAP password used to bind to the LDAP server.

- **init_ldap_cache**
  Attribute name for: This enables caching of the user, group, and LDAP policy data in the client to reduce the turnaround of similar LDAP transactions.

- **init_ldap_host**
  Attribute name for: The LDAP server host name.

- **init_ldap_max_search_size**
  Attribute name for: Set the maximum number of entries returned from an LDAP search.

- **init_ldap_port**
  Attribute name for: The LDAP server host port (a numerical string).

- **init_ldap_prefer_rw_svr**
  Attribute name for: This makes the writable LDAP server the preferred one for the client.

- **init_ldap_replica**
  Attribute name for: The LDAP server replica's.

- **init_ldap_ssl_keyfile**
  Attribute name for: The LDAP server's SSL keyfile.

- **init_ldap_ssl_keyfile_dn**
  Attribute name for: The LDAP server's SSL keyfile distinguished name.

- **init_ldap_ssl_keyfile_pwd**
  Attribute name for: The LDAP server's SSL keyfile password.
- **init_ldap_use_compare**
  Attribute name for: Chooses whether the client will use the ldap_compare or the ldap_bind call to authenticate the user's password.

- **init_listen_flags**
  Attribute name for: Flags to enable the reception of policy cache update notifications.

- **init_max_handle_groups**
  Attribute name for: The max number of handle groups (4k handles in each) permitted.

- **init_mode**
  Attribute name for: Azn API authzn engine initialization mode.

- **init_namespace_location**
  Attribute name for: CDS namespace location for exporting the RPC endpoints for local policy cache updates.

- **init_pac_svc**
  PAC service definition attribute.

- **init_qop**
  Attribute name for: Quality of protection for communications with IVAcld.

- **init_remote_ns_loc**
  Attribute name for: Optional namespace location for ivacld binding information.

- **init_set_perminfo_attrs**
  Attribute name for: The set of attributes that the caller is interested in receiving from the Azn.decision_access_allowed_ext method in the permission info attribute list.

- **init_tcp_port**
  Attribute name for: The TCP port upon which policy cache updates should be received.

- **init_udp_port**
  Attribute name for: The UDP port upon which policy cache updates should be received.
Chapter 2: Java Wrappers Reference

- **IV_DCE**
  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**
  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.

- **IV_UNAUTH**
  Mechanism ID for an unauthenticated 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_UARAF**
  Mechanism ID for a UARAF 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_add**
  Operation string for add.

- **operation_attach**
  Operation string for attach.

- **operation_browse**
  Operation string for browse.

- **operation_bypasstod**
  Operation string for bypasstod.

- **operation_connect**
  Operation string for connect.

- **operation_control**
  Operation string for control.

- **operation_create**
  Operation string for create.

- **operation_delegation**
  Operation string for delegation.

- **operation_delete**
  Operation string for delete.
- **operation_execute**
  Operation string for execute.
- **operation_forward**
  Operation string for forward.
- **operation_list_directory**
  Operation string for list directory.
- **operation_modify**
  Operation string for modify.
- **operation_password**
  Operation string for password.
- **operation_read**
  Operation string for read.
- **operation_server_admin**
  Operation string for server admin.
- **operation_traverse**
  Operation string for traverse.
- **operation_view**
  Operation string for view.
- **perminfo_al**
  Audit level that determines the audit events that are performed for this authorization check.
- **perminfo_al_admin**
- **perminfo_al_all**
- **perminfo_al_deny**
- **perminfo_al_error**
- **perminfo_al_none**
  Azn.perminfo_al values
- **perminfo_al_permit**
- **perminfo_all_attrs**
  This value is passed in the azn_init_set_perminfo_attrs attribute to azn_initialize() to specify that the caller wished to receive all attributes that can potentially be returned by the authzn engine on an extended authzn decision call.
Chapter 2: Java Wrappers Reference

- **perminfo_qop**
  The quality of protection required between caller and the domain before access to the protected object can be granted.

- **perminfo_qop_uint**
  This is the same as the attribute above except it is returned as an unsigned int.

- **perminfo_qop_uint_integrity**
- **perminfo_qop_uint_non**
  Azn.perminfo_qop_uint values

- **perminfo_qop_uint_privacy**
- **perminfo_wm**
  Warning mode.

- **perminfo_wm_permitted**
  Access permitted because of warning mode.

- **PERMITTED**
  Decision result value for a permitted operation.

- **svc_version**
  Used to return the version string of a service module to the Authorization API runtime.

**Constructor Index**

- **Azn**

**Method Index**

- **attrlist_add_entry**
  Add an attribute name and its string value to an Attribute List.

- **attrlist_add_entry_buffer**
  Add an attribute name and its binary value to an Attribute List.

- **attrlist_copy**
  Copy an Attribute List.

- **attrlist_create**
  Create a new empty Attribute List.

- **attrlist_delete**
  Delete an Attribute List.
- **attrlist_delete_entry**
  Delete an entry from an Attribute List.

- **attrlist_get_entry_buffer_value**
  Get a binary value associated with an attribute name in the specified Attribute List.

- **attrlist_get_entry_string_value**
  Get a string value associated with an attribute name in the specified Attribute List.

- **attrlist_get_names**
  Get the set of all attribute names in the specified Attribute List.

- **attrlist_name_get_num**
  Get the number of values associated with an attribute name in the specified Attribute List.

- **creds_combine**
  Combine two credentials and return the resulting combined credential.

- **creds_compare**
  Compare two credentials.

- **creds_copy**
  Copy a credentials structure.

- **creds_create**
  Create a new empty credentials structure.

- **creds_delete**
  Delete an existing credentials structure.

- **creds_for_subject**
  Obtain a specified credentials structure from a combined credentials chain.

- **creds_get_attrlist_for_subject**
  Obtain attribute information from a specified credentials structure.

- **creds_get_pac**
  Create a privilege attribute certificate (PAC) by invoking a specified PAC service on the supplied credentials.
Chapter 2: Java Wrappers Reference

- **creds_modify**
  Modify an existing credential and return a handle to a new credential containing the modifications.

- **creds_num_ofSubjects**
  Get the number of individual credentials in a credentials chain.

- **decision_access_allowed**
  Obtain the result of an access control decision.

- **decision_access_allowed_ext**
  Obtain the result of an access control decision using application specific context information and return the reason why the decision was made.

- **entitlement_get_entitlements**
  Obtain entitlements from the specified entitlements service plug-in.

- **error_code**
  Build a complete error code from a major and minor status.

- **error_major**
  Obtain the major error code associated with a status code that was returned by one of the methods in this class.

- **error_minor**
  Obtain the minor error code associated with a status code that was returned by one of the methods in this class.

- **error_minor_get_string**
  Obtain the message string describing the implementation specific minor error code.

- **id_get_creds**
  Obtain the handle to the credentials associated by a specified authorization authority with the indicated authorization ID.

- **initialize**
  Initialize the authorization service.

- **pac_get_creds**
  Obtain a handle to new credentials that are derived from a privilege attribute certificate (PAC) by a specified PAC service.

- **setDebugMode**
  Sets the debug mode for the native method implementation.
- **shutdown**
  Cleans up internal authorization service state in preparation for shutdown.

- **util_client_authenticate**
  Perform authentication for a username and password.

- **util_errcode**
  Returns the Azn error status code generated from the specified major and minor error codes.

- **util_handle_is_valid**
  Test the given handle for validity.

- **util_password_authenticate**
  Perform authentication for a username and password, and return the authentication information when authentication is successful.

- **util_server_authenticate**
  Perform authentication from a keytab file and start a background thread to refresh the login context as necessary.
### Variables

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cred_version</td>
<td>The credential version</td>
</tr>
<tr>
<td>public static String</td>
<td></td>
</tr>
<tr>
<td>cred_mech_id</td>
<td>The mechanism ID for this cred</td>
</tr>
<tr>
<td>public static String</td>
<td></td>
</tr>
<tr>
<td>cred_principal_uuid</td>
<td>The UUID of the entity</td>
</tr>
<tr>
<td>public static String</td>
<td></td>
</tr>
<tr>
<td>cred_principal_name</td>
<td>The string name of the entity</td>
</tr>
<tr>
<td>public static String</td>
<td></td>
</tr>
<tr>
<td>cred_group_uuids</td>
<td>The string group UUID memberships of this entity</td>
</tr>
<tr>
<td>public static String</td>
<td></td>
</tr>
<tr>
<td>Method</td>
<td>Signature</td>
</tr>
<tr>
<td>------------------------</td>
<td>------------------------------------</td>
</tr>
<tr>
<td>cred_group_names</td>
<td>public static String cred_group_names</td>
</tr>
<tr>
<td>cred_dce_name</td>
<td>public static String cred_dce_name</td>
</tr>
<tr>
<td>cred_ldap_dn</td>
<td>public static String cred_ldap_dn</td>
</tr>
<tr>
<td>cred_uraf_name</td>
<td>public static String cred_uraf_name</td>
</tr>
<tr>
<td>cred_user_info</td>
<td>public static String cred_user_info</td>
</tr>
<tr>
<td>cred_auth_method</td>
<td>public static String cred_auth_method</td>
</tr>
</tbody>
</table>
cred_authnmech_info

public static String cred_authnmech_info

Remarks

Any authentication mechanism info passed in the mech info struct

cred_ip_address

public static String cred_ip_address

Remarks

The IP address information passed in the mech info struct

cred_qop_info

public static String cred_qop_info

Remarks

The Quality of Protection information passed in the mech info struct
One of “none”, “integrity” or “privacy”.

cred_browser_info

public static String cred_browser_info

Remarks

The browser information passed in the mech info struct

perminfo_all_attrs

public static String perminfo_all_attrs

Remarks

This value is passed in the azn_init_set_perminfo_attrs attribute to azn_initialize() to specify that the caller wished to receive all attributes that can potentially be returned by the authzn engine on an extended authzn decision call. This will override any other attributes passed in the azn_init_set_perminfo_attrs attribute.
**perminfo_qop**

```java
public static String perminfo_qop
```

**Remarks**

The quality of protection required between caller and the domain before access to the protected object can be granted. The QOP value is always returned and will be one of “none”, “integrity” or “privacy”.

**perminfo_qop_uint**

```java
public static String perminfo_qop_uint
```

**Remarks**

This is the same as the attribute above except it is returned as an unsigned int. It is always equal to one of the values defined below.

**perminfo_qop_uint_none**

```java
public static int perminfo_qop_uint_none
```

Azn.perminfo_qop_uint values

**perminfo_qop_uint_integrity**

```java
public static int perminfo_qop_uint_integrity
```

**perminfo_qop_uint_privacy**

```java
public static int perminfo_qop_uint_privacy
```

**perminfo_wm**

```java
public static String perminfo_wm
```

Warning mode. When warning mode is enabled access is always granted. If access should not have been granted then the access is logged.
Chapter 2: Java Wrappers Reference

**perminfo_wm_permitted**

```java
class perminfo_wm_permitted {
    public static String perminfo_wm_permitted
}
```

**Remarks**

Access permitted because of warning mode. The boolean indicator is used to tell the caller that access was granted because warning mode is enabled.

---

**perminfo_al**

```java
class perminfo_al {
    public static String perminfo_al
}
```

**Remarks**

Audit level that determines the audit events that are performed for this authorization check.

---

**perminfo_al_none**

```java
class perminfo_al_none {
    public static int perminfo_al_none
}
```

**Remarks**

Azn.perminfo_al values

---

**perminfo_al_permit**

```java
class perminfo_al_permit {
    public static int perminfo_al_permit
}
```

---

**perminfo_al_deny**

```java
class perminfo_al_deny {
    public static int perminfo_al_deny
}
```

---

**perminfo_al_error**

```java
class perminfo_al_error {
    public static int perminfo_al_error
}
```

---

**perminfo_al_admin**

```java
class perminfo_al_admin {
    public static int perminfo_al_admin
}
```
#### perminfo_al_all

```java
public static int perminfo_al_all
```

**Remarks**

Used to return the version string of a service module to the Authorization API runtime.

#### svc_version

```java
public static String svc_version
```

**Remarks**

Used to return the version string of a service module to the Authorization API runtime.

#### pac_svc_dce_epac

```java
public static String pac_svc_dce_epac
```

**Remarks**

PAC Service ID: DCE EPAC service. Converts a DCE EPAC structure into an azn API cred. This service does not perform the reverse operation of building a DCE EPAC from an azn API cred. This service is only for use with azn_pac_get_creds(). This service takes one parameter: `-loadnames`. If this parameter is passed to the service as an argument in the service definition then the EPAC PAC service will attempt to load the real string names of the principal and group uuids into the returned Authorization API credential. This enables the return of values in the azn_cred_group_names attribute for creds generated from DCE EPACs. This feature is turned off by default.
ent_svc_pd_pobj

public static String ent_svc_pd_pobj

Remarks
Entitlements Service ID: Policy Director Protected Object
Entitlements. This service takes a credential, a directory within the
protected object tree and a set of operations and returns the list of
protected objects in the given directory and its subdirectories for
which the given Authorization API credential has permission to
perform the requested set of operations. The output is returned as a
multi-string valued attribute. NOTE: this is a default service. That is,
when a null entitlements service ID is specified to
azn_entitlements_get_entitlements() then this is the service that is
called.

ent_svc_pd_pobj_path

public static String ent_svc_pd_pobj_path

Remarks
Input attribute: the protected object directory path to be searched.

ent_svc_pd_pobj_reqd_ops

public static String ent_svc_pd_pobj_reqd_ops

Remarks
Input attribute: the requested set of operations for the credential.

ent_svc_pd_pobj_matches

public static String ent_svc_pd_pobj_matches

Remarks
Output attribute: the list of objects which match the input criteria.
**PERMITTED**

```java
public static final int PERMITTED

Remarks
Decision result value for a permitted operation.
```

**NOT_PERMITTED**

```java
public static final int NOT_PERMITTED

Remarks
Decision result value for a not permitted operation.
```

**IV_UNAUTH**

```java
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**

```java
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**

```java
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.
```
**Chapter 2: Java Wrappers Reference**

**IV_URAF**

```java
public static String IV_URAF
```

**Remarks**

Mechanism ID for a URAF 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**

```java
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_bypasstod**

```java
public static String operation_bypasstod
```

**Remarks**

Operation string for `bypasstod`.

**operation_browse**

```java
public static String operation_browse
```

**Remarks**

Operation string for `browse`. 

Version 3.7
operation_control
public static String operation_control

Remarks
Operation string for control.

operation_traverse
public static String operation_traverse

Remarks
Operation string for traverse.

operation_delegation
public static String operation_delegation

Remarks
Operation string for delegation.

operation_view
public static String operation_view

Remarks
Operation string for view.

operation_modify
public static String operation_modify

Remarks
Operation string for modify.

operation_delete
public static String operation_delete

Remarks
Operation string for delete.
Chapter 2: Java Wrappers Reference

operation_server_admin

public static String operation_server_admin

Remarks
Operation string for server admin.

operation_audit

public static String operation_audit

Remarks
Operation string for audit.

operation_integrity

public static String operation_integrity

Remarks
Operation string for integrity.

operation_list_directory

public static String operation_list_directory

Remarks
Operation string for list directory.

operation_password

public static String operation_password

Remarks
Operation string for password.

operation_create

public static String operation_create

Remarks
Operation string for create.
operation_add
public static String operation_add
   Remarks
   Operation string for add.

operation_privacy
public static String operation_privacy
   Remarks
   Operation string for privacy.

operation_read
public static String operation_read
   Remarks
   Operation string for read.

operation_execute
public static String operation_execute
   Remarks
   Operation string for execute.

operation_list_directory
public static String operation_list_directory
   Remarks
   Operation string for list directory.

operation_connect
public static String operation_connect
   Remarks
   Operation string for connect.
Chapter 2: Java Wrappers Reference

operation_forward

public static String operation_forward

Remarks
Operation string for forward.

init_mode

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

public static String init_qop

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

init_cfg_file

public static String init_cfg_file

Remarks
Attribute name for: Fully qualified Configuration file name.
init_max_handle_groups

public static String init_max_handle_groups

Remarks
Attribute name for: The max number of handle groups (4k handles in each) permitted.

init_set_perminfo_attrs

public static String init_set_perminfo_attrs

Remarks
Attribute name for: The set of attributes that the caller is interested in receiving from the Azn.decision_access_allowed_ext method in the permission info attribute list. By default there is no information returned by this call. Adding a value to this attribute at init time enables the attribute to be returned as permission info if it is applicable to the current decision call. This list may also include attributes that are user defined. For example, by setting an attribute on an ACL or POP using “modify set attribute” command of “pdadmin”. The following attribute constants are recognized by the authzn engine. Refer below for more information on these permission info attributes: Azn.perminfo_all_attrs Return all attributes. Azn.perminfo_al Audit level (uint). Azn.perminfo_qop Quality of Protection (string). Azn.perminfo_qop_uint Quality of Protection (uint). Azn.perminfo_wm Warning mode (bool). Azn.perminfo_wm_permitted Access permitted by warning mode (bool).

init_db_file

public static String init_db_file

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

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

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

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_tcp_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

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**

public static String init_tcp_port

Remarks

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

**init_udp_port**

public static String init_udp_port

Remarks

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

**init_ldap_bind_dn**

public static String init_ldap_bind_dn

Remarks

Attribute name for: The LDAP distinguished name used to bind to the LDAP server.

**init_ldap_bind_pwd**

public static String init_ldap_bind_pwd

Remarks

Attribute name for: The LDAP password used to bind to the LDAP server.

**init_ldap_ssl_keyfile**

public static String init_ldap_ssl_keyfile

Remarks

Attribute name for: The LDAP server's SSL keyfile.
**init_ldap_host**

```java
public static String init_ldap_host
```

**Remarks**

Attribute name for the LDAP server host name.

**init_ldap_port**

```java
public static String init_ldap_port
```

**Remarks**

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

**init_ldap_ssl_keyfile**

```java
public static String init_ldap_ssl_keyfile
```

**Remarks**

Attribute name for the LDAP server's SSL keyfile.

**init_ldap_ssl_keyfile_dn**

```java
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**

```java
public static String init_ldap_ssl_keyfile_pwd
```

**Remarks**

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

code
public static String init_ldap_replica

Remarks
Attribute name for: The LDAP server replica’s. Each string value is of the format <ldap-server>, <port>, <type>, <pref> where:
<ldap-server> is the network name of the server, <port> is the port on the ldap server, <type> is one of “readonly” or “readwrite” and <pref> is a level from 1 to 10. 10 is the highest preference.

init_ldap_cache

code
public static String init_ldap_cache

Remarks
Attribute name for: This enables caching of the user, group, and LDAP policy data in the client to reduce the turnaround of similar LDAP transactions. The string value can be “true” or “false”. Defaults to “false”.

init_ldap_prefer_rw_svr

code
public static String init_ldap_prefer_rw_svr

Remarks
Attribute name for: This makes the writable LDAP server the preferred one for the client. Calls will still failover to a read-only server if the write server is offline. The string value can be “true” or “false”. Defaults to “false”.

init_ldap_max_search_size

code
public static String init_ldap_max_search_size

Remarks
Attribute name for: Set the maximum number of entries returned from an LDAP search. This is further constrained by the limits set by the LDAP server.
Chapter 2: Java Wrappers Reference

**init_ldap_use_compare**

```java
public static String init_ldap_use_compare
```

**Remarks**

Attribute name for: Chooses whether the client will use the ldap_compare or the ldap_bind call to authenticate a the users password. This applies to calls to azn_util_client_authenticate() and azn_util_password_authenticate() with an LDAP registry. The string value can be “true” or “false”. Default is “false” which is to use the ldap_bind call.

**init_ent_svc**

```java
public static String init_ent_svc
```

**Remarks**

Entitlements service definition attribute.

**init_pac_svc**

```java
public static String init_pac_svc
```

**Remarks**

PAC service definition attribute.

**init_cred_mod_svc**

```java
public static String init_cred_mod_svc
```

**Remarks**

Credential modification service definition attribute.

**init_remote_ns_loc**

```java
public static String init_remote_ns_loc
```

**Remarks**

Attribute name for: Optional namespace location for ivacld binding information. The typical default value is /./subsys/intraverse/authzn/active
Constructor

Azn

Azn()

Remarks

Constructor.

Methods

attrlist_add_entry

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

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

Parameters

Input
  attr_list
    AznAttrList object.
  attr_name
    Name attribute of the entry to be added.
  string_value
    Value (string) attribute of the entry to be added.

Description

This function 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

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

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

Parameters

    Input
    attr_list
        AznAttrList object.
    attr_name
        Name attribute of the entry to be added.
    buffer_value
        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**

`public static native int attrlist_create(AznAttrList attr_list)`

Creates an attribute list.

**Parameters**

**Input/Output**

`attr_list`  
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**

`public static native int attrlist_delete(AznAttrList attr_list)`

Deletes an attribute list.

**Parameters**

**Input/Output**

`attr_list`  
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_delete_entry

public static native int attrlist_delete_entry(AznAttrList attr_list, String attr_name)

Remarks
Delete an entry from an Attribute List. The attr_list object passed to this method is both an input and output parameter. The attribute name and its values in the Attribute List will be delete from the attr_list object.

Parameters
attr_list
AznAttrList object.
attr_name
Attribute name.

Returns
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_copy

public static native AznAttrList attrlist_copy(AznAttrList attr_list)

Remarks
Copy an Attribute List. A copy of the attr_list object passed to this method is returned.

Parameters
attr_list
AznAttrList object.

Returns
A copy of the input AznAttrList object.
attrlist_get_entry_buffer_value

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

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

**Parameters**

**Input**

attr_list
AznAttrList object

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

value_index
Index within the entry of the string attribute value to be returned.

**Input /Output**

buffer_value
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

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

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

**Parameters**

**Input**

- `attr_list`<br>    AznAttrList object
- `attr_name`<br>    Name attribute of the entry from which the value attribute is to be returned.
- `value_index`<br>    Index within the entry of the string attribute value to be returned.

**Input /Output**

- `string_value`<br>    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

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

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

Parameters

    Input
    attr_list
        AznAttrList object

    Output
    attr_names
        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

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

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

Parameters

Input

attr_list
AznAttrList object

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

Input/Output

num_values
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

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

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

**Parameters**

- **Input**
  - `creds`
    - AznCreds object for the credentials chain whose first indexed entry is the credential of the initiator of the request.
  - `creds_to_add`
    - AznCreds object for the credentials to be added to an existing credentials chain.

- **Output**
  - `combined_creds`
    - 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 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_copy

public static native AznCreds creds_copy(AznCreds creds)

Copy a credentials structure. A copy of the creds object passed to this method is returned.

Parameters
creds
AznCreds object.

Returns
A copy of the input AznCreds object is returned.

creds_compare

public static native boolean creds_compare(AznCreds cred1,
AznCreds cred2)

Compare two credentials.

Parameters
creds
AznCreds object.

Returns
true if the credentials are equal, false otherwise.
creds_create

public static native int creds_create(AznCreds creds)

Creates a new, empty credentials chain.

Parameters

Input /Output
creds
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

public static native int creds_delete(AznCreds creds)

Deletes a credentials chain.

Parameters

Input /Output
creds
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

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

Obtains a specified credentials chain from a combined credentials chain.

**Parameters**

**Input**

combined_creds

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

subject_index

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).

**Input/Output**

new_creds

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

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

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

**Parameters**

**Input**
- `creds`
  AznCreds object representing a credentials chain which contains one or more individual credentials structures.
- `subject_index`
  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).

**Input /Output**
- `creds_attrlist`
  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

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

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

Parameters

Input
creds
AznCreds object for the credentials whose information is used
to build the PAC.
pac_svc_id
Identification (id) of the PAC service that produces the PAC.

Input /Output
pac
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

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

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

**Parameters**

**Input**

- `creds`
  
  AznCreds object for the credentials to be modified.

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

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

**Input /Output**

- `new_creds`
  
  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

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

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

**Parameters**

**Input**

creds

AznCreds object for the credentials chain.

**Input /Output**

num_of_subjects

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

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

    Makes an access control decision.

Parameters

Input
creds
AznCreds object for the initiator's credential chain.

protected_resource
Name of the target resource of the request.

operation
Name of the requested operation.

Input /Output
permission
AznInteger object where the decision result is returned. If the returned status value is Azn.S_COMPLETE, the returned permission will be Azn.PERMISSION or Azn.NOT_PERMISSION.

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.PERMISSION or Azn.NOT_PERMISSION. 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.
**Chapter 2: Java Wrappers Reference**

---

### decision_access_allowed_ext

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

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

**Parameters**

**Input**

- **creds**
  
  AznCreds object for the initiator's credentials chain.

- **protected_resource**
  
  Name of the target of the request.

- **operation**
  
  Name of the requested operation.

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

**Input/Output**

- **permission**
  
  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**
  
  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
Remarks

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.
Chapter 2: Java Wrappers Reference

entitlement_get_entitlements

public static native int entitlement_get_entitlements(
    AznCreds creds,
    String entitlements_svc_id,
    AznAttrList app_context,
    AznAttrList entitlements)

Obtain entitlements from the specified entitlements service plug-in.

Parameters

creds
    The credentials of the user for whom the calling application
    needs a list of applicable entitlements.

tenentitlements_svc_id
    The service identification for the Entitlements Service plug-in.

app_context
    The requested action and the protected resource for which the
    user's entitlements are needed.

tenentitlements
    Attributes list to contain the user's entitlements. The
    entitlements are returned as a list of name-value pairs.

Returns

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

error_code

public static native int error_code(int major, int minor)

Build a complete error code from a major and minor status. This is
the reciprocal of calling the error_major and error_minor calls from
the result returned from other auth api functions.

Parameters

major
    Major error code for the returned status code.

minor
    Minor error code for the returned status code.

Returns

An Azn status code.
error_major

public static native int error_major(int azn_status)

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

Parameters

Input

azn_status

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

public static native int error_minor(int azn_status)

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.

Parameters

Input

azn_status

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

public static native int error_minor(int azn_status, AznString 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.

Parameters

Input

azn_status

An Azn status code.

Output

string

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

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

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

Parameters

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

Input /Output
    new_creds
        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 Authorization API.
Chapter 2: Java Wrappers Reference

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

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

Initializes the authorization service.

Parameters

Input

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

Input /Output

init_info
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 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**

```java
class AznCreds {
    // Implementation
}
```

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

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

**Parameters**

**Input**

- `pac`:
  - AznBuffer object that holds the supplied PAC.

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

**Output**

- `new_creds`:
  - 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.
Chapter 2: Java Wrappers Reference

**setDebugMode**

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

Sets the debug mode for the native method implementation.

**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**

**Input**

- `mode`
  
  Debug mode.

**shutdown**

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

Cleans up internal authorization service state in preparation for 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**

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

Performs authentication from a user name and password.

**Parameters**

**Input**

- `principal_name`
  
  Name of the principal (user) to be authenticated.

- `password`
  
  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_errcode**

```java
public static native int util_errcode(int major, int minor)
```

Returns the `Azn` error status code generated from the specified major and minor error codes.

**Parameters**

- `major`
  
  Major component of the error status code.

- `minor`
  
  Minor component of the error status code.

**Returns**

An `Azn` error status code.
util_handle_is_valid

public static native boolean util_handle_is_valid(long handle)

Test the given handle for validity. This is used to determine whether
a given handle actually references real data. Can be used by
Authorization API and Authorization API service programmers to
test a handle passed to it before attempting to work with it.

Parameters
handle
The handle to test for validity.

Returns
ture if the handle is valid or false if invalid.

util_password_authenticate

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

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

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

password
Password for the user.

Input /Output
mechanism_id
AznString object set with the mechanism ID identifying the
authentication mechanism.

authinfo
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

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

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

Parameters

Input

principal_name
    Name of the user (principal) to be authenticated.

keytab_path
    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

- handle
  Attribute list handle

Constructor Index

- AznAttrList
  A constructor for an AznAttrList which initializes the attribute list handle to 0.
- AznAttrList(long)
  A constructor for an AznAttrList object which takes the Attribute List handle as a parameter.

Variable

handle

public long handle

Remarks

Attribute list handle.
Chapter 2: Java Wrappers Reference

Constructors

**AznAttrList**

public AznAttrList()

*Remarks*

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

**AznAttrList(long)**

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:

<table>
<thead>
<tr>
<th>C API Data Type</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>azn_authdce_t</td>
<td>For DCE credentials</td>
</tr>
<tr>
<td>azn_authldap_t</td>
<td>For LDAP credentials</td>
</tr>
<tr>
<td>azn_unauth_t</td>
<td>For unauthenticated</td>
</tr>
</tbody>
</table>

Variable Index

- user_identity
  DCE principal name or LDAP distinguished name
- auth_method
  Authentication method identification for DCE or LDAP credentials
- ipaddr
  IP address of the user that sent the request.
- qop
  Quality of protection level
- user_info
  Optional user information.
- browser_info
  Optional browser information.
- authmech_info
  Optional authentication information; not used for unauthenticated credentials.
Constructor Index

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

Variables

---

**user_identity**

```
public String user_identity
```

**Remarks**

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

---

**auth_method**

```
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**

```
public long ipaddr
```

**Remarks**

IP address of requesting user.

---

**qop**

```
public String qop
```

**Remarks**

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

public String user_info

Remarks
Additional user information that might be required for auditing.

browser_info

public String browser_info

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

authnmech_info

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

public AznAuthInfo()

Remarks
Constructor for an AznAuthInfo object which initializes all the data members to 0 or null.
Class `com.ibm.pd.Authzn.AznBuffer`

```java
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**

- **value**
  
  The byte array containing the buffer value.

**Constructor Index**

- **AznBuffer**

**Variable**

```java
value
public byte value[]
```

**Remarks**

The byte array containing the buffer value.

**Constructor**

```java
AznBuffer
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 azncreds_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
  Credentials structure handle

Constructor Index

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

- AznCreds(long)
  Constructor for an AznCreds object which takes the credentials structure handle as a parameter.

Variable

handle

public long handle

Remarks

Credentials structure handle.
### Constructors

**AznCreds**

```java
public AznCreds()
```

**Remarks**

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

**AznCreds(long)**

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

**Remarks**

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

**Parameters**

**Output**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>handle</td>
<td>Credentials structure handle.</td>
<td></td>
</tr>
</tbody>
</table>
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

- value

  Integer value.

Constructor Index

- AznInteger

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

Variable

value

public int value

Remarks

Integer value.

Constructor

AznInteger

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

- value
  String value.

Constructor Index

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

Variable

value

    public String value

    Remarks

    String value.

Constructor

AznString

    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

- value
  
  Array of string values.

Constructor Index

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

Variable

```
value
```

public String value[]

Remarks

Array of string values.

Constructor

```
AznStrings
```

public AznStrings()

Remarks

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