IBM Security Access Manager for Web
Version 7.0

Administration Java Classes Developer Reference

SC23-6514-02

IBM Security Access Manager for Web
Version 7.0

Administration Java Classes Developer Reference

SC23-6514-02

Note
Before using this information and the product it supports, read the information in "Notices" on page 141.

Edition notice

Note: This edition applies to version 7, release 0, modification 0 of IBM Security Access Manager (product number 5724-C87) and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

Figures

Tables

About this publication

IBM Security Access Manager for Web, formerly called IBM Tivoli Access Manager for e-business, is a user authentication, authorization, and web single sign-on solution for enforcing security policies over a wide range of web and application resources.

This reference contains information about how to use Security Access Manager administration Java™ classes and methods to enable an application to programmatically perform Security Access Manager administration tasks. This document describes the Java implementation of the Security Access Manager administration API. See the *IBM Security Access Manager for Web: Administration C API Developer Reference* for information regarding the C implementation of these APIs.

Information about the `pdadmin` command-line interface (CLI) can be found in the *IBM Security Access Manager for Web: Command Reference*.

Intended audience

This reference is for application programmers writing programs in and Java programming language to administer the users and objects associated with the Security Access Manager product.

Readers must be familiar with:

- Microsoft Windows and UNIX operating systems
- Database architecture and concepts
- Security management
- Internet protocols, including HTTP, TCP/IP, File Transfer Protocol (FTP), and Telnet
- The user registry that Security Access Manager is configured to use
- Lightweight Directory Access Protocol (LDAP) and directory services, if used by your user registry
- Authentication and authorization

To enable Secure Sockets Layer (SSL) communication, you must be familiar with SSL protocol, key exchange (public and private), digital signatures, cryptographic algorithms, and certificate authorities.

Access to publications and terminology

This section provides:

- A list of publications in the “IBM Security Access Manager for Web library” on page x.
- Links to “Online publications” on page xi.
- A link to the “IBM Terminology website” on page xii.

IBM Security Access Manager for Web library

The following documents are in the IBM Security Access Manager for Web library:

- *IBM Security Access Manager for Web Quick Start Guide*, GI11-9333-01

  Provides steps that summarize major installation and configuration tasks.
- *IBM Security Web Gateway Appliance Quick Start Guide* – Hardware Offering

  Guides users through the process of connecting and completing the initial configuration of the WebSEAL Hardware Appliance, SC22-5434-00
- *IBM Security Web Gateway Appliance Quick Start Guide* – Virtual Offering

  Guides users through the process of connecting and completing the initial configuration of the WebSEAL Virtual Appliance.
- *IBM Security Access Manager for Web Installation Guide*, GC23-6502-02

  Explains how to install and configure Security Access Manager.
- *IBM Security Access Manager for Web Upgrade Guide*, SC23-6503-02

  Provides information for users to upgrade from version 6.0, or 6.1.x to version 7.0.
- *IBM Security Access Manager for Web Administration Guide*, SC23-6504-02

  Describes the concepts and procedures for using Security Access Manager. Provides instructions for performing tasks from the Web Portal Manager interface and by using the **pdadmin** utility.
- *IBM Security Access Manager for Web WebSEAL Administration Guide*, SC23-6505-02

  Provides background material, administrative procedures, and reference information for using WebSEAL to manage the resources of your secure Web domain.
- *IBM Security Access Manager for Web Plug-in for Web Servers Administration Guide*, SC23-6507-02

  Provides procedures and reference information for securing your Web domain by using a Web server plug-in.
- *IBM Security Access Manager for Web Shared Session Management Administration Guide*, SC23-6509-02

  Provides administrative considerations and operational instructions for the session management server.
- *IBM Security Access Manager for Web Shared Session Management Deployment Guide*, SC22-5431-00

  Provides deployment considerations for the session management server.
- *IBM Security Web Gateway Appliance Administration Guide*, SC22-5432-00

  Provides administrative procedures and technical reference information for the WebSEAL Appliance.
- *IBM Security Web Gateway Appliance Configuration Guide for Web Reverse Proxy*, SC22-5433-00

  Provides configuration procedures and technical reference information for the WebSEAL Appliance.
- *IBM Security Web Gateway Appliance Web Reverse Proxy Stanza Reference*, SC27-4442-00

  Provides a complete stanza reference for the IBM® Security Web Gateway Appliance Web Reverse Proxy.
- *IBM Security Access Manager for Web WebSEAL Configuration Stanza Reference*, SC27-4443-00

  Provides a complete stanza reference for WebSEAL.

- *IBM Global Security Kit: CapiCmd Users Guide*, SC22-5459-00

  Provides instructions on creating key databases, public-private key pairs, and certificate requests.
- *IBM Security Access Manager for Web Auditing Guide*, SC23-6511-02

  Provides information about configuring and managing audit events by using the native Security Access Manager approach and the Common Auditing and Reporting Service. You can also find information about installing and configuring the Common Auditing and Reporting Service. Use this service for generating and viewing operational reports.
- *IBM Security Access Manager for Web Command Reference*, SC23-6512-02

  Provides reference information about the commands, utilities, and scripts that are provided with Security Access Manager.
- *IBM Security Access Manager for Web Administration C API Developer Reference*, SC23-6513-02

  Provides reference information about using the C language implementation of the administration API to enable an application to perform Security Access Manager administration tasks.
- *IBM Security Access Manager for Web Administration Java Classes Developer Reference*, SC23-6514-02

  Provides reference information about using the Java language implementation of the administration API to enable an application to perform Security Access Manager administration tasks.
- *IBM Security Access Manager for Web Authorization C API Developer Reference*, SC23-6515-02

  Provides reference information about using the C language implementation of the authorization API to enable an application to use Security Access Manager security.
- *IBM Security Access Manager for Web Authorization Java Classes Developer Reference*, SC23-6516-02

  Provides reference information about using the Java language implementation of the authorization API to enable an application to use Security Access Manager security.
- *IBM Security Access Manager for Web Web Security Developer Reference*, SC23-6517-02

  Provides programming and reference information for developing authentication modules.
- *IBM Security Access Manager for Web Error Message Reference*, GI11-8157-02

  Provides explanations and corrective actions for the messages and return code.
- *IBM Security Access Manager for Web Troubleshooting Guide*, GC27-2717-01

  Provides problem determination information.
- *IBM Security Access Manager for Web Performance Tuning Guide*, SC23-6518-02

  Provides performance tuning information for an environment that consists of Security Access Manager with the IBM Tivoli Directory Server as the user registry.

Online publications

IBM posts product publications when the product is released and when the publications are updated at the following locations:

IBM Security Access Manager for Web Information Center
The http://pic.dhe.ibm.com/infocenter/tivihelp/v2r1/topic/com.ibm.isam.doc_70/welcome.html site displays the information center welcome page for this product.

IBM Publications Center
The http://www-05.ibm.com/e-business/linkweb/publications/servlet/pbi.wss site offers customized search functions to help you find all the IBM publications that you need.

IBM Terminology website

The IBM Terminology website consolidates terminology for product libraries in one location. You can access the Terminology website at http://www.ibm.com/software/globalization/terminology.

Related publications

This section lists the IBM products that are related to and included with the Security Access Manager solution.

Note: The following middleware products are not packaged with IBM Security Web Gateway Appliance.

IBM Global Security Kit

Security Access Manager provides data encryption by using Global Security Kit (GSKit) version 8.0.x. GSKit is included on the *IBM Security Access Manager for Web Version 7.0* product image or DVD for your particular platform.

GSKit version 8 includes the command-line tool for key management, GSKCapiCmd (**gsk8capicmd_64**).

GSKit version 8 no longer includes the key management utility, iKeyman (**gskikm.jar**). iKeyman is packaged with IBM Java version 6 or later and is now a pure Java application with no dependency on the native GSKit runtime. Do not move or remove the bundled *java*/jre/lib/gskikm.jar library.

The *IBM Developer Kit and Runtime Environment, Java Technology Edition, Version 6 and 7, iKeyman User's Guide for version 8.0* is available on the Security Access Manager Information Center. You can also find this document directly at:

http://download.boulder.ibm.com/ibmdl/pub/software/dw/jdk/security/60/iKeyman.8.User.Guide.pdf

Note:

GSKit version 8 includes important changes made to the implementation of Transport Layer Security required to remediate security issues.

The GSKit version 8 changes comply with the Internet Engineering Task Force (IETF) Request for Comments (RFC) requirements. However, it is not compatible with earlier versions of GSKit. Any component that communicates with Security Access Manager that uses GSKit must be upgraded to use GSKit version 7.0.4.42, or 8.0.14.26 or later. Otherwise, communication problems might occur.

IBM Tivoli Directory Server

IBM Tivoli Directory Server version 6.3 FP17 (6.3.0.17-ISS-ITDS-FP0017) is included on the *IBM Security Access Manager for Web Version 7.0* product image or DVD for your particular platform.

You can find more information about Tivoli Directory Server at:

http://www.ibm.com/software/tivoli/products/directory-server/

IBM Tivoli Directory Integrator

IBM Tivoli Directory Integrator version 7.1.1 is included on the *IBM Tivoli Directory Integrator Identity Edition V 7.1.1 for Multiplatform* product image or DVD for your particular platform.

You can find more information about IBM Tivoli Directory Integrator at:

http://www.ibm.com/software/tivoli/products/directory-integrator/

IBM DB2 Universal Database™

IBM DB2 Universal Database Enterprise Server Edition, version 9.7 FP4 is provided on the *IBM Security Access Manager for Web Version 7.0* product image or DVD for your particular platform. You can install DB2® with the Tivoli Directory Server software, or as a stand-alone product. DB2 is required when you use Tivoli Directory Server or z/OS® LDAP servers as the user registry for Security Access Manager. For z/OS LDAP servers, you must separately purchase DB2.

You can find more information about DB2 at:

http://www.ibm.com/software/data/db2

IBM WebSphere® products

The installation packages for WebSphere Application Server Network Deployment, version 8.0, and WebSphere eXtreme Scale, version 8.5.0.1, are included with Security Access Manager version 7.0. WebSphere eXtreme Scale is required only when you use the Session Management Server (SMS) component.

WebSphere Application Server enables the support of the following applications:

- Web Portal Manager interface, which administers Security Access Manager.
- Web Administration Tool, which administers Tivoli Directory Server.
- Common Auditing and Reporting Service, which processes and reports on audit events.
- Session Management Server, which manages shared session in a Web security server environment.
- Attribute Retrieval Service.

You can find more information about WebSphere Application Server at:

http://www.ibm.com/software/webservers/appserv/was/library/

Accessibility

Accessibility features help users with a physical disability, such as restricted mobility or limited vision, to use software products successfully. With this product, you can use assistive technologies to hear and navigate the interface. You can also use the keyboard instead of the mouse to operate all features of the graphical user interface.

Visit the IBM Accessibility Center for more information about IBM's commitment to accessibility.

Technical training

For technical training information, see the following IBM Education website at http://www.ibm.com/software/tivoli/education.

Support information

IBM Support provides assistance with code-related problems and routine, short duration installation or usage questions. You can directly access the IBM Software Support site at http://www.ibm.com/software/support/probsub.html.

The *IBM Security Access Manager for Web Troubleshooting Guide* provides details about:

- What information to collect before you contact IBM Support.
- The various methods for contacting IBM Support.
- How to use IBM Support Assistant.
- Instructions and problem-determination resources to isolate and fix the problem yourself.

Note: The **Community and Support** tab on the product information center can provide more support resources.

Chapter 1. Introduction to the administration API

The Security Access Manager Java runtime component includes the Java language version of the Security Access Manager administration API. The Security Access Manager Java runtime component provides a set of Java classes and methods for the administration of selected Security Access Manager administration objects. These classes and methods provide a way for applications to administer users, groups, protected objects, and access control lists.

You can use the Security Access Manager application developer kit (ADK) to enable your application to programmatically administer Security Access Manager administration objects.

This chapter contains the following topics:

Note: If you are familiar with the C language interface to the Security Access Manager administration API, see Appendix A, "Differences between the C and Java administration API," on page 63 for a general overview of differences. A mapping of C APIs to Java classes and methods can be found in Appendix C, "Administration API equivalents," on page 67.

Administration Java classes overview

A set of Java classes is provided for creating, modifying, examining, listing, and deleting each of the preceding object types. The classes include the methods necessary for manipulating each of these administration objects. These administration Java classes are packaged in the `PD.jar` file that is installed as part of the Security Access Manager Java runtime environment component. Applications which use the Java runtime environment that is provided with Security Access Manager automatically have access to these classes and methods.

The administration Java classes can be used to administer the following types of objects:

- Policies
- Users
- Groups
- Access control lists (ACLs)
- Extended ACL actions
- Protected object policies (POPs)
- Protected objects
- Protected object spaces
- Authorization rules
- Domains
- Web, or single signon (SSO), resources

- Web resource groups
- Resource credentials

The administration API Java classes communicate directly with the Security Access Manager policy server component. The API establishes an authenticated, Secure Socket Layer (SSL) session with the Security Access Manager policy server process. After the SSL session is established, the classes can send administration requests to the policy server.

The Security Access Manager policy server component services these requests in the same manner that it would service any other incoming requests.

System administrators also can use the **pdadmin** command-line interface to accomplish Security Access Manager administration tasks. The Java administration classes and methods map closely to these commands. Appendix C, "Administration API equivalents," on page 67 describes the commands that match Java administration API methods. Some Java methods do not have a **pdadmin** command-line equivalent.

Note: Do not use the **svrsslcfg** command-line interface with the Java applications. Use the `SvrSslCfg` Java class to provide this feature.

Accessing the Javadoc HTML documentation

This section explains where to access the Javadoc HTML documentation so you can add authorization and security services to Java application.

To add IBM Security Access Manager for Web authorization and security services to new or existing Java applications, use the Javadoc information provided with the Security Access Manager application developer kit (ADK) along with this book.

Consult the Javadoc HTML documentation for deprecated Java APIs before you update existing IBM Security Access Manager for Web application.

Copy the Javadoc HTML information stored in the entire `AM_BASE/nls/javadocs` directory tree, to another location on your development system. Then, uninstall the Security Access Manager ADK and runtime components. Only the IBM Security Access Manager Runtime for Java component is necessary for running Java applications. See Table 1 on page 3 for the Javadoc installation location.

Other ways to manipulate administration objects

You can use the Java administration APIs to manipulate administration objects. Other than that, you also can use the pdadmin command-line interface, Administration C API, and Registry Direct Java API to manipulate administration objects.

pdadmin command-line interface (CLI)
The **pdadmin** command-line interface is explained in the *IBM Security Access Manager for Web: Command Reference*.

Administration C API
The administration C API provides support for these administration objects. Refer to the *IBM Security Access Manager for Web: Administration C API Developer Reference* for details.

Registry Direct Java API

The Registry Direct API directly accesses the underlying Security Access Manager registry rather than through Authorization servers or Policy Server. The API also provides access to most of the underlying registry user attributes and the attributes available through the traditional Security Access Manager Java API. See the "Registry Direct Java API" on page 84 section for details.

Java administration API components

The administration API consists of the following components:

- The administration Java classes
- Javadoc information for the associated Java classes and methods
- A demonstration application

The administration API Java classes are distributed in the Security Access Manager Java runtime component for each platform. The remaining administration API components are distributed in the Security Access Manager Application Developer Kit component.

Application development kit

The Javadoc information associated with the administration Java classes, methods, and examples are in the Security Access Manager application developer kit (ADK) component package.

Table 1 lists the files that are installed as part of the Security Access Manager ADK component. The `PD.jar` file, even though it is installed as part of the Security Access Manager Java runtime component, is listed in the table for completeness.

Table 1. Administration API application development kit files

Directory	Files	File description
AM_BASE`/nls/javadocs/pdjrte/index.html`	`index.html` (and many others)	Javadoc HTML documentation for the Java classes and methods provided with the Security Access Manager Java runtime component.
JAVA_HOME`/lib/ext`	`PD.jar`	The Java Archive (JAR) file which contains the classes and methods associated with the administration APIs. **Note:** When you use the `pdjrtecfg` command-line interface to configure the Security Access Manager Java runtime component to a particular JRE, this archive file is copied to *JAVA_HOME*`/lib/ext`. You do not have to modify the CLASSPATH in your environment to access the classes and methods defined in this archive file. For WebSphere 8 JRE, PD.jar is copied in `WAS_HOME/tivoli/tam`. If you use the WebSphere 8 JRE stand-alone, or outside of running server profile JVM, manually add it to your CLASSPATH.

Table 1. Administration API application development kit files (continued)

Directory	Files	File description
AM_BASE/example/pdadminapi_demo/java	README.PDAdminDemo PDAdminDemo.java PDAdminDemo.class PDAdminDemo$ConsoleEraser.class	A demonstration program is provided to illustrate the use of the administration Java APIs. Copy the demonstration program to any other directory. The readme file explains how to run and recompile the demonstration program.
AM_BASE/example/authz_demo/java	PDCallbackHandler.class PDDemoSetup.class PDDemoSetup.java PDJaasDemo$1.class PDJaasDemo.class PDJaasDemo.java PDListObjectsDemo.class PDListObjectsDemo.java PDPermissionDemo.class PDPermissionDemo.java README.JaznDemo	These files consist of various demonstrations which illustrate the use of Security Access Manager's Java authorization APIs. See the README.JaznDemo for a description on how to run the various demonstrations.
AM_BASE/example/local_remote_demo/java	PDLRAuthzDemo1.class PDLRAuthzDemo1.java PDLRAuthzDemo2$1.class PDLRAuthzDemo2$2.class PDLRAuthzDemo2.class PDLRAuthzDemo2.java PDLRExerciseDialog$1.class PDLRExerciseDialog$2.class PDLRExerciseDialog$3.class PDLRExerciseDialog$4.class PDLRExerciseDialog.class PDLRExerciseDialog.java PDLRTestDemo.class PDLRTestDemo.java PDtamdemoException.class PDtamdemoException.java PDTimer.class PDTimer.java README.PDLocalRemoteDemo	These files consist of a demonstration that illustrates the use of both the local and remote modes of administration and authorization APIs. The demonstration provides a graphical user interface for defining the various setup parameters. See the README.PDlocalRemoteDemo for a description on how to generate the documentation for the demonstration classes.

Building Java applications with the administration API

To develop Java applications that use the Security Access Manager administration API, you must install and configure the required software.

Security Access Manager software requirements

You must install and configure a secure domain. If you do not have secure domain installed, install one before beginning application development.

A minimum installation consists of a single system with the following Security Access Manager components installed:

- Security Access Manager runtime environment (see the note about the installation prerequisite)
- Security Access Manager Java runtime component
- Security Access Manager policy server
- Security Access Manager ADK

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

- Security Access Manager runtime environment (see the note about the installation prerequisite)
- Security Access Manager Java runtime component
- Security Access Manager ADK

For Security Access Manager installation instructions, see the corresponding section in the *IBM Security Access Manager for Web: Installation Guide* for your operating system platform.

Note: The installation of Security Access Manager requires the installation of the Security Access Manager runtime component. This runtime component is not required for developing or deploying Java applications. In this specific situation, you can reclaim the disk space that is used by the Security Access Manager ADK and runtime components while saving the Javadoc HTML information and the example files from the ADK component.

To reclaim this disk space, copy the Javadoc information, consisting of the entire *AM_BASE*`/nls/javadocs` directory tree, and copy the sample Java program, in the *AM_BASE*`/example` directory tree, to another location on your development system and then uninstall the Security Access Manager ADK and runtime components.

Configuration of the Java runtime component to a particular Java runtime environment

Configure the IBM Security Access Manager Runtime for Java component to use the proper JRE on the system by using the **pdjrtecfg** command. The Security Access Manager Java runtime component can be configured to several different JREs on the same system, if required. See the *IBM Security Access Manager for Web: Installation Guide* for details.

Configuration of the Java administration classes

The `com.tivoli.pd.jcfg.SvrSslCfg` Java class must be used to configure the administration Java APIs. See the *IBM Security Access Manager for Web: Authorization Java Classes Developer Reference* for details on the `SvrSslCfg` utility.

Note:

1. Do not use the **svrsslcfg** command-line interface to create configuration files that are to be used with Java applications.
2. The `com.tivoli.mts.SvrSslCfg` class provided in previous versions of Security Access Manager and IBM SecureWay Policy Director has been deprecated. Use the new `com.tivoli.pd.jcfg.SvrSslCfg` class instead.

Security requirements

To run a Java application in the context of a Java security manager, the application must have proper Java permissions to use the administration Java APIs. If the application is not installed as a Java extension in the *JAVA_HOME*`/lib/ext` directory, an entry must be added to the *JAVA_HOME*`/lib/security/java.policy` file.

To grant the necessary permission to the Java applications located in the `/sb/pdsb/export/classes` directory, and all its subdirectories, the necessary Java permissions to use authorization Java classes and methods, add a statement like the following to the `java.policy` file:

```
// Give applications in /sb/pdsb/export/classes and
// its subdirectories access to the Access Manager
// Administration APIs
grant codeBase "file:/sb/pdsb/export/classes/-" {
permission javax.security.auth.AuthPermission "PDAdmin";
};
```

Figure 1. Granting Java permission to applications

Invoke administration Java classes and methods from a privileged block, `doPrivileged()`, to alleviate the need for the application callers to have this Java permission as well.

The `PD.jar` file is signed, but verification of the signing of JAR files is not supported in this version of Security Access Manager.

Java administration API example program

The Security Access Manager ADK includes the complete Java source code for an example program that demonstrates the use of the administration Java classes.

The example program demonstrates how to perform the following tasks:

- Initialize an administration API security context
- Display an error message
- Create a new Security Access Manager user
- Set a user account to be valid
- Create a new group
- Add the new user to the group
- Delete a group
- Delete a user

Deployment of a Java administration API application

Java applications that have been developed using the Security Access Manager administration API must be run on systems that are configured as part of a Security Access Manager secure domain.

To run an administration Java application, you must have installed the Security Access Manager Java runtime component.

Note: Information about installing the Security Access Manager Java runtime component can be found in the *IBM Security Access Manager for Web: Installation Guide*.

Gathering of problem determination information

Security Access Manager components can be configured to log information to one or more trace files. You can enable tracing for the policy server, the authorization server, the Java runtime component, or any system using the Security Access Manager runtime environment.

Enabling tracing on the policy server

Procedure

1. Edit the `/etc/pdmgrd_routing` file, located in the installation directory for the Security Access Manager policy server.
2. Uncomment the last line.
3. Shut down and restart the policy server daemon **pdmgrd**.

Enabling tracing on the authorization server

Edit the `/etc/pdacld_routing` file to enable tracing on the authorization server

Procedure

1. Open the `/etc/pdacld_routing` file, in the installation directory for the Security Access Manager authorization server.

   Note: `pdacld_routing` applies to the default authorization server. If multiple instances of the authorization server exist, each routing file is prefixed with the instance name. For example, if an authorization server instance name is `instance1`, the routing file becomes `instance1-pdacld_routing`.
2. Uncomment the last line.
3. Shut down and restart the authorization server daemon, **pdacld**.

Enabling tracing in the Java runtime component

Enable tracing in the Java runtime component by editing the properties file settings that the `com.tivoli.pd.jcfg.SvrSslCfg` command creates.

About this task

Tracing for the Security Access Manager Java runtime component is controlled by the properties file settings the `com.tivoli.pd.jcfg.SvrSslCfg` command create.

Procedure

1. Edit the properties file created.
2. Update the line associated with the required *application-server-name* to set `isLogging` to `true`:

   `baseGroup.PDJ`*`application-server-name`*`TraceLogger.isLogging=true`

   Each Java application can be configured to use a different properties file, and the properties file can have any name and be located in any directory.

   The `PDJLog.properties` file, located in the `PolicyDirector` subdirectory of the associated JRE, is installed by the Security Access Manager Java runtime environment component. This properties file is associated with, and can be used to enable tracing in, the `pdjrtecfg` command as well as the `com.tivoli.pd.jcfg.SvrSslCfg` command.

Gathering of message logs

Message logs associated with applications that are configured bu using the `com.tivoli.pd.jcfg.SvrSslCfg` command are, by default, which is written to a set of three files: `msg__application_name1.log`, `msg__application_name2.log`, and `msg__application_name3.log`, where *application_name* is the name that is specified with the **appSvr** parameter of `SvrSslCfg`.

Each file is 512 KB in size, and the `msg__application_name1.log` file always contains the latest messages.

The number, size, and base name of these files can be configured by using the options in the configuration file.

Note: There are two underscore characters (_) following the characters `msg` in the default file names.

The `PDJLog.properties` file controls the message logging for Java programs that are not configured with the `com.tivoli.pd.jcfg.SvrSslCfg` command. This properties file specifies different file names for each class of Security Access Manager messages: FATAL, ERROR, WARNING, NOTICE, or NOTICEVERBOSE. Each class of message is written to a set of three files, with names of the following form:

```
msg__amj_fatalN.log
msg__amj_errorN.log
msg__amj_warningN.log
msg__amj_noticeN.log
msg__amj_noticeverboseN.log
```

For more information about message logging, see the *IBM Security Access Manager for Web: Troubleshooting Guide*.

Gathering of trace logs

Trace logs associated with applications that are configured by using the `com.tivoli.pd.jcfg.SvrSslCfg` command are, by default, which is written to a set of three files: `trace__application_name1.log`, `trace__application_name2.log`, and `trace__application_name3.log`, where *application_name* is the name that is specified with the **appSvr** parameter of `SvrSslCfg`.

Each file is 512 KB in size, and the `trace__application_name1.log` file always contains the latest trace entries.

The number, size, and base name of these files can be configured by using the options in the configuration file.

Note: There are two underscore characters (_) following the characters `trace` in the default file names.

The `PDJLog.properties` file controls the trace logging for Java programs that are not configured with the `com.tivoli.pd.jcfg.SvrSslCfg` command. By default, this trace output is directed to a set of three files that are called `trace__amj1.log`, `trace__amj2.log`, and `trace__amj3.log`. The number, size, and base name of these files can be configured by using the options in the `PDJLog.properties` file.

For more information, see the *IBM Security Access Manager for Web: Troubleshooting Guide*.

Chapter 2. Using the administration API

Each Java application that uses the administration API must perform certain tasks necessary for API initialization, shut down, and error handling. The administration API provides methods for each of these tasks.

The following sections in this chapter describe the supported functions:

Note: If you are familiar with the administration C API described in the *IBM Security Access Manager for Web: Administration C API Developer Reference*, see Appendix A, "Differences between the C and Java administration API," on page 63.

Administration objects

Each Security Access Manager administration object that can be manipulated directly from a Java application is represented by a corresponding Java class. This section describes the administration objects.

The following objects are supported in this version of Security Access Manager:

PDAdmin
: This class is used to initialize and shut down the operations associated with using the Security Access Manager administration classes and methods. The methods in this class are applicable to all administration objects.

PDAuthzRule
: This class represents a Security Access Manager authorization rule.

PDContext
: This class encapsulates the information needed to establish a communication session between the Java application and the Security Access Manager policy server. Both user ID and password-based and certificate-based authentication are supported by this class. Multiple `PDContext` objects can be created and used within the same Java virtual machine (JVM).

  `PDContext` creation is a resource exhaustive operation. Although there is no upper limit to creating multiple `PDContext` objects, system resource limitation eventually determines how many can be successfully created and used. Create and pool only few `PDContext` objects in the application environment. Reuse the small number of created `PDContext` objects whenever possible within the same application.

Because each user application needs are different, pooling `PDContext` objects is not mandatory. Pool `PDContext` objects if you have a server application that makes numerous calls. If you have an application that makes only an occasional call, or if you have various stand-alone applications which make calls, pooling is not necessary.

PDDomain
This class represents a Security Access Manager policy server domain.

PDUser
This class represents a user in the Security Access Manager policy server.

PDGroup
This class represents a group in the Security Access Manager policy server.

PDPolicy
This class represents the policy information that is associated with a particular Security Access Manager user or, in the case of the global policy, that is associated with all users. The **PDPolicy** class is used to set and retrieve account policy information from the user registry on a global or per-user basis.

PDAcl This class represents an access control list (ACL), which in turn consists of a list of ACL entries.

PDAclEntry
This class represents an entry in an ACL.

PDAclEntryUser
This class represents a user ACL entry and controls access for a particular user.

PDAclEntryGroup
This class represents a group ACL entry and controls access for all members in a group.

PDAclEntryAnyOther
This class represents the **any-other**, or **any-other authenticated**, entry in an ACL. This ACL entry applies to any user who is authenticated into the Security Access Manager secure domain but is not included in a separate user or group ACL entry.

PDAclEntryUnAuth
This class represents the **unauthenticated** user ACL entry. This ACL entry is applied to any user who was not authenticated by Security Access Manager.

PDProtObject
This class represents a protected object. A protected object represents a resource that is to be protected, and it has an ACL associated with it. Each protected object is uniquely identified by an ID.

PDProtObjectSpace
This class represents the protected object space object. An object space is a logical grouping of protected objects which represents a set of related resources to be protected. Each object space is uniquely identified by an ID.

PDPop
This class represents a protected object policy, or POP, which can be attached to a `PDProtObject` object.

PDAdmSvcPobj
This class represents the value of a Security Access Manager administration service protected object.

PDAction
This class represents a permission.

PDActionGroup
This class represents a collection of PDAction objects.

PDRgyGroupName
This class represents the name of a Security Access Manager group in the underlying user registry.

PDRgyUserName
This class represents the name of a Security Access Manager user in the underlying user registry.

PDRgyName
This class represents the name of a Security Access Manager object in the underlying user registry. This object is either a Security Access Manager user name or group name.

PDAppSvrSpecLocal
This class represents configuration information for a local Java application server.

PDAppSvrSpecRemote
This class represents configuration information for a remote Java application server.

PDSvrInfo
This class represents a Security Access Manager policy server or authorization server and is used when creating or changing the configuration for a Java application server.

PDAppSvrInfo
This class represents a read-only view of a Java application server configuration information.

PDServer
This class represents a Security Access Manager policy server, authorization server, or other application server.

PDSSOResource
This class represents a single sign-on (SSO) resource.

PDSSOResourceGroup
This class represents a single sign-on (SSO) resource group.

PDSSOCred.CredID
This class represents the credential identification information for each member of the list returned by the `PDSSOCred.listSSOCreds` method.

PDSSOCred.CredInfo
This class represents the credential information for each member of the list returned by the `PDSSOCred.listAndShowSSOCreds` method.

PDException
This class creates an exception to reflect that an error or other exceptional condition occurred.

PDMessage
This class represents a single Security Access Manager message and includes the message code, severity, and the localized message text.

PDMessages
This class represents a list of one or more Security Access Manager messages.

The methods associated with these classes are threadsafe.

Common classes

This section describes class used for both administration and authorization methods.

PDAttrs
This class represents a list of Security Access Manager attributes.

PDAttrValue
This class represents the value of a Security Access Manager attribute.

PDAttrValues
This class represents a collection of values for a particular attribute that is unordered and that does not allow duplicates.

PDAttrValueList
This class represents a collection of values for a particular attribute that is ordered and allows duplicates.

Initializing the administration API

Before using the administration API in a Java application, you must initialize the `PDAdmin` object.

This initialization is accomplished by calling the `PDAdmin.initialize()` method, by passing the name of the application and a `PDMessages` object. Messages are described in more detail in "Messages" on page 18.

See the following sample administration API initialization:

```
PDMessages messages = new PDMessages();

PDAdmin.initialize("myApplicationName", messages);
```

Establishing a security context

After initializing the administration API, you must create an SSL connection between the Java application and the Security Access Manager policy server.

This connection is referred to as a *security context* by the administration API. The security context provides for the secure transfer of administrative requests and data between the Java application and the policy server.

A security context can be established using either user ID and password-based authentication or certificate-based authentication. In either case, the security context is represented by the `PDContext` object. Multiple `PDContext` objects can be created and used within the same JVM.

`PDContext` creation is a resource exhaustive operation. Although there is no upper limit to creating multiple `PDContext` objects, system resource limitation eventually determines how many can be successfully created and used. Create and pool only a few `PDContext` objects in the application environment.

For more Information about Java authentication classes and methods, see the *IBM Security Access Manager for Web: Authorization Java Classes Developer Reference*.

User ID and password-based authentication

This section describes the information you need to establish a security context with user ID and password authentication.

To establish a security context using user ID and password-based authentication, you need the following information:

admin user ID
A Security Access Manager user ID with the appropriate administrative authority, such as `sec_master`.

admin password
The password associated with the administrator user ID.

locale The locale that is to be used for returning message data to the application. When this value is not supplied as a key parameter, the `PDContext` constructor uses the default locale.

domain
The Security Access Manager policy server domain to which the user is authenticated. When this value is not supplied, the domain is obtained from the configuration file URL. When the configuration file URL does not contain domain information, the local domain associated with the Java Runtime Environment is used.

configuration file URL
The uniform resource locator (URL) to the configuration file created by the Java **`SvrSslCfg`** class. The URL must use the `file:///` format.

Note: Do not use the `svrsslcfg` command-line interface to create a configuration file that is to be used by a Java application.

To create the security context, create a `PDContext` object as shown in Figure 2 on page 14.

```
// Create locale for US English

Locale myLocale = new Locale("ENGLISH", "US");

/*
Create a security context using our locale. Need to supply a user ID with
administrative privileges in Access Manager (like sec_master) along with
its password and a URL of the form file:/// to the configuration file created
by the SvrSslCfg class.
*/

PDContext myContext = new PDContext(myLocale,
adminName,
adminPassword,
domain,
configFileURL);
```

Figure 2. Creating a security context using user ID and password-based authentication

The contents of the configuration file created by the Java `SvrSslCfg` class is not externalized and is subject to change without notice. Users must not use the information in the configuration file directly.

Certificate-based authentication

This section describes the information you need to establish a security context with certificate-based authentication.

To establish a security context using certificate-based authentication, you need the following information:

locale The locale that is to be used for returning message data to the application.

configuration file URL
The URL to the configuration file created by the Java `SvrSslCfg` class. The URL must use the `file:///` format.

Note: Do not use the **svrsslcfg** command-line interface to create a configuration file that is to be used by a Java application.

To create the security context, create a `PDContext` object as shown in Figure 3.

```
// Create locale for US English

Locale myLocale = new Locale("ENGLISH", "US");

/*
Create a security context using certificate-based authentication.
The URL to the configuration file must use the file:/// format. The
configuration file is created by the SvrSslCfg class.
*/

PDContext myContext = new PDContext(myLocale,
configFileURL);
```

Figure 3. Creating a security context using certificate-based authentication

The contents of the configuration file created by the Java `SvrSslCfg` class is not externalized and is subject to change without notice. Users must not use the information in the configuration file directly.

Manipulating administration objects

Each Java class, which represents an administration object, provides static methods to create, list, modify, and delete objects stored on the Security Access Manager policy server.

Changes to administration objects on the policy server are immediately available to other applications.

The constructor of each class can be used to obtain a local copy of a specific administration object. You can use the class instance methods for the following purposes:

- To retrieve data from the local object.
- To modify both the local copy of the object and the object stored on the policy server.

Use static methods for command-line and batch-oriented applications by using the administration API. For interactive applications, use instance methods.

Creating objects

You can use the administration API to create Security Access Manager objects necessary to complete administrative tasks.

Before you create an object, you must initialize the administration API and establish a security context.

To create an object, use the static creation method associated with the administration object. For example, to create a Security Access Manager user, you would use the `PDUser.createUser()` static method. This method results in the Security Access Manager user being created immediately on the policy server. See the following static method sample, Creating a user.

Creating a user

```
/*--------------------------------------------------------------------
* Create a user, using the PDUser.createUser() static method, and
* assign the user to a specific group. This method sends a
* request to the policy server to create the user.
*--------------------------------------------------------------------
*/

// Set up all of the user's attributes
String name = "Stephanie Luser";
String firstName = "Stephanie";
String lastName = "Luser";
String password = "herpassword";
String description = "Descriptive text for Stephanie Luser";
String rgyName = "cn=" + name + "," + rgySuffix;
PDRgyUserName pdRgyUserName =
new PDRgyUserName(rgyName, firstName, lastName);
boolean ssoUser = false;
boolean pwdPolicy = true;
ArrayList groupList = new ArrayList();
groupList.add(groupAdministrativeAssistants);
messages.clear();

PDUser.createUser(mySecurityContext,
name,
pdRgyUserName,
description,
password.toCharArray(),
groupList,
ssoUser,
pwdPolicy,
messages);
```

Obtaining a local copy of an object

To obtain a local copy of an administration object, use the constructor for the Java class that represents the administration object.

For example, to get a copy of the `PDUser` object representing a particular Security Access Manager user, you would use the `PDUser` constructor as shown in Figure 4.

```
/*--------------------------------------------------------------------
* Obtain a user using the PDUser constructor.
*--------------------------------------------------------------------
*/

// Set up all of the user's attributes
String name = "Zachary Wommbat";
String firstName = "Zachary";
String lastName = "Wommbat";
String rgyName = "cn=" + name + "," + rgySuffix;
PDRgyUserName pdRgyUserName =
new PDRgyUserName(rgyName, firstName, lastName);
messages.clear()

PDUser user = new PDUser(mySecurityContext,
pdRgyUserName,
messages);
```

Figure 4. Getting a local copy of a `PDUser` object

After obtaining a local copy of the administration object, use the instance methods on the object to retrieve or set data associated with the object.

Note: You can obtain the local copy of the administration object and use command-line interface, the administration C API or Java class methods to change the object on the policy server. A few instance methods can detect inconsistencies between data in the local object and data in the policy server, but other instances lack this capability. Ensure that changes made to administration objects are consistent and predictable while using the instance methods.

Reading object values

You can view administration object data by using the instance methods associated with the administration object.

To use the instance methods, first obtain a local copy of the object, as outlined in "Obtaining a local copy of an object" on page 16. After obtaining the object, retrieve information about the object by using the instance methods. For example, to get the description associated with a Security Access Manager user from a local copy of the `PDUser` object:

```
userDescription = user.getDescription();
```

Setting object values

You can change administration object data with the instance methods associated with the administration object. You can also change administration object data with the static methods associated with the Java class representing the administration object.

Before using the instance methods, you must obtain a local copy of the object, as outlined in "Obtaining a local copy of an object" on page 16. After obtaining the object, you can change information about the object using the instance methods. For example, to disable the account associated with a Security Access Manager user from a local copy of the `PDUser` object, use:

```
user.setAccountValid(mySecurityContext,
                     false,   // Disable the account
                     messages);
```

The instance method changes both local copy of the administration object as well as the object stored on the policy server.

To update the `PDUser` object on the policy server, use the static method:

```
PDUser.setAccountValid(mySecurityContext,
                       name,
                       false,   // Disable the account
                       messages);
```

Listing objects

Some administrative tasks require the Java application to obtain a list of objects. This section describes the method for listing objects based on their Java class.

For example, an administrator must review the list of existing users to decide whether a new user must be created.

Table 2 on page 18 lists the appropriate method to use to list objects based on the Java class that represents an administration object.

Table 2. Methods used to list objects

Object	Method to list objects
PDAcl	PDAcl.listAcls
PDGroup	PDGroup.listGroups
PDProtObject	PDProtObject.listProtObjects PDProtObject.listProtObjectsByAcl
PDProtObjectSpace	PDProtObjectSpace.listProtObjectSpaces
PDUser	PDUser.listUsers
PDDomain	PDDomain.listDomains
PDAuthzRule	PDAuthzRule.listAuthzRules

Deleting objects

To delete an object, use the static deletion method associated with the administration object.

For example, to delete a Security Access Manager user, use the `PDUser.deleteUser()` static method. This method deletes the Security Access Manager user from the policy server immediately. See the following sample static method, Deleting a user.

Deleting a user

```
/*-------------------------------------------------------------------
* Delete a user
*-------------------------------------------------------------------
*/

// Set up all of the user's attributes
String name = "Leah Allen";
messages.clear();

PDUser.deleteUser(mySecurityContext,
name,
true,
messages);
```

Messages

All constructors, static methods, and instance methods have an output parameter consisting of a `PDMessages` object. This section describes the attributes of a `PDMessages` object.

In addition, exceptions generated by Security Access Manager contain a `PDMessages` object.

A `PDMessages` object can be empty or contain one or more `PDMessages` objects. Each `PDMessages` object represents a single message and consists of the following attributes:

Message code
A hexadecimal number that uniquely identifies the message.

Message text
The localized text of the message.

Severity

An indication of the severity of the message:

- Informational
- Warning
- Error

The message text is localized based on the `PDContext` object used when the method is invoked except in the case of read-only instance method on a local administration object.

When a method completes successfully, check the `PDMessages` object for any informational or warning messages associated with the action performed. If an error is encountered during processing, a `PDException` exception is thrown, which might have messages associated with it.

The same `PDMessages` object can be used on multiple method invocations. Use the **clear()** method to clear the contents of the `PDMessages` object between method invocations.

The *IBM Security Access Manager for Web: Error Message Reference* contains a list of the messages issued by Security Access Manager along with an explanation of the message and the suggested corrective action. Messages are indexed by hexadecimal and decimal message number, as well as by message identifier.

Handling errors

All constructors, instance methods, and static methods generate a `PDException` exception when an error or unexpected event occurs. This exception contains a `PDMessages` object that contains one or more `PDMessages` objects.

For more information about messages and message handling, see "Messages" on page 18.

A `PDException` object also might contain a wrapped exception thrown by another Java component. Information about this wrapped exception can be obtained using the methods of the `PDException` object.

The *IBM Security Access Manager for Web: Error Message Reference* contains a list of the messages issued by Security Access Manager along with an explanation of the message and the suggested corrective action.

Shutting down the administration API

After using the administration API, you must shut down the `PDAdmin` object.

This shutdown is accomplished by calling the `PDAdmin.shutdown()` method as shown in the following example:

Shutting down the administration API

```
PDAdmin.shutdown(messages);
```

Character-based data considerations

Character-based data, such as user IDs and passwords, are stored and manipulated as strings of Unicode characters by the Java classes and methods.

This character data is converted from Unicode into UTF-8 (Universal Character Set Transformation Format-8) before being sent to the Security Access Manager policy server and stored in the user registry. Similarly, data from the user registry and the policy server is received in UTF-8 and converted into Unicode. Unicode and UTF-8 both allow any character in any locale to be uniquely represented.

PDContext application design considerations

This section explains some aspects of PDContext application design to consider.

Note:

For detailed information about PDContext pooling class, see the Javadoc HTML documentation. For details in accessing the Javadoc HTML, see the Table 1 on page 3 section.

There are a few points to consider before you use the PDContext class to design applications, such as:

- How many applications use `PDContext` concurrently?
- How many users per application use `PDContext`?
- Will the application initiate administrative requests to the Policy Server? If yes, is it handled by a single user or many users?

The most important concept to remember when you design an application that uses the PDContext class is that `PDContext` objects must be reused whenever possible. Do not create a `PDContext` object for every operation. Doing so quickly exhaust the sessions available on the server.

For every `PDContext` object created by your application, a session is created and maintained on the Policy Server. The validity of the session is controlled by the `ssl-v3-timeout` parameter in the `pd.conf` file. The default value for `ssl-v3-timeout` in the`pd.conf` file is 7200 seconds (2 hours).

Each `PDContext` object can manage several connections. However, only one connection is active per context at a time. Other connections are queued up and blocked until the current connection is completed, allowing the next connection to complete. By default each `PDContext` object supports 10 connections. The connection support is advantageous in multithreaded applications which allow multiple threads to easily share a `PDContext`. This approach is best for application with few threads and only occasional requests to the policy server.

The number of `PDContext` objects that are created represents the number of concurrent requests that are handled by your application. For applications that require many threads and frequent requests to the policy server, the `PDContext` objects must be maintained in a pool where `PDContext` objects are checked out as needed and returned when the operation completes. Depending on the number of threads and the number `PDContext` objects in the pool, the approach yields good performance results.

Application requirements vary considerably depending on the goal. As a result, we can provide only general guidelines on a `PDContext` Pool design. Basic requirements include:

- Maintain the pool as a private class variable.
- All methods that access the pool must be synchronized.
- Start with a pool size of 10 `PDContext` objects and adjust as needed to improve performance.
- Creating `PDContext` objects can take a lot of time. Avoid the use of large pool sizes.
- To improve overall performance, do not create `PDContext` objects immediately. Initialize the pool when the application starts, instead.

Chapter 3. Administering users and groups

The administration API provides a collection of classes and methods for administering Security Access Manager users and groups. This section describes the tasks that those classes and methods accomplish.

Information about Security Access Manager users and groups is stored in the user registry. You can use the administration API to both modify and access user and group settings in the user registry. In addition, the administration API provides classes and methods to administer password and account policy settings both on a per user and global basis.

Security Access Manager provides the **pdadmin** command-line interface (CLI) that accomplishes many of the same user, group, and policy administration tasks. Application developers who have previously used the **pdadmin** command to manage a Security Access Manager secure domain finds the administration API functions straightforward to implement.

This chapter contains the following topics:

Administering users

The administration API provides classes and methods for creating, accessing, listing, and deleting Security Access Manager user information within the user registry.

The name of a user is not case-sensitive. The following user names all refer to the same Security Access Manager user:

- `user`
- `USER`
- `User`
- `UsEr`

The `PDUser.createUser` method creates a user in the user registry that is used by the Security Access Manager policy server.

Note: When a user definition exists in the user registry, use the `PDUser.importUser` method instead.

The `PDUser.importUser` method imports an existing user definition from your user registry into Security Access Manager and allows the user definition to be managed by Security Access Manager.

Use the `PDUser.deleteUser` method to delete a user from Security Access Manager.

Table 3 lists the user administration functions.

User registry difference: Leading and trailing blank spaces in a user name do not make the name unique when using an LDAP or Active Directory user registry. However, leading and trailing blanks do make the user name unique when using a Domino server as a user registry. To keep name processing consistent regardless of what user registry is being used, do not define user names with leading or trailing blanks.

Table 3. Administering users

Method	Description
PDUser.createUser	Creates the specified user.
PDUser.importUser	Creates a Security Access Manager user by importing an existing user from your user registry.
PDUser.deleteUser	Deletes the specified user.
PDUser.listUsers	Lists Security Access Manager users.

For detailed reference information about these methods, see the Javadoc HTML documentation. For information about accessing this documentation, see Table 1 on page 3.

Administering user information

The administration API allows you to administer the information associated with a Security Access Manager user.

When a user account has been created in the user registry, you can set and get various information about the user. You must create a security context between the calling application and the Security Access Manager policy server before you can access the user registry. You can obtain the user registry information for a user object by specifying either the Security Access Manager user name or the user registry name.

Table 4 lists the methods available for administering user information.

Table 4. Administering user information

Methods	Description
PDUser constructor	Instantiates a user object for the specified Security Access Manager or user registry name.
PDUser object.getDescription	Returns the user description.
PDUser object.getRgyName	Returns the user registry name for the user.
PDUser object.getId	Returns the name of the object.
PDUser object.getFirstName	Returns the first-name attribute for the user.

Table 4. Administering user information (continued)

Methods	Description
PDUser object.getLastName	Returns the last-name attribute for the user.
PDUSer object.getLastLogin	Returns the last login time.
PDUser object.getPolicy	Returns the password and account policy settings associated with the user.
PDUser object.getGroups	Lists the groups in which the user is a member.
PDUser object.isAccountValid	Returns the account-valid indicator for the user.
PDUser object.isPDUser	Returns an indicator whether the user is a Security Access Manager user.
PDUser object.isSSOUser	Returns an indicator whether the user has single sign-on capabilities.
PDUser.setDescription *PDUser object*.set Description	Sets a user description.
PDUser.setAccountValid *PDUser object*.setAccountValid	Enables or disables a user account.
PDUser.setSSOUser *PDUser object*.setSSOUser	Enables or disables the single signon capabilities of a user.
PDUser object.isPasswordValid	Returns the enabled indicator for the user password.
PDUser.setPassword *PDUser object*.setPassword	Sets a user password.
PDUser.setPasswordValid *PDUser object*.setPasswordValid	Enables or disables a user password.

For detailed reference information about these methods, see the Javadoc HTML documentation. For information about accessing this documentation, see Table 1 on page 3.

Administering user account policies

You can manage user access by setting account policies. You can specify policies that apply either only to a single user or for all users.

When a user account policy attribute is set to a value and enforced, the value always takes precedence over a value that is set for the general policy. This is true even if the value that is set for the general policy is more restrictive.

If an account policy attribute for a user is not enforced, then the value is set for the general policy. If that value is set and enforced, then the value is applied for the user.

Table 5 describes the administration API methods that are used to modify or access account policies.

Table 5. Administering user account policies

Method	Description
PDUser.getUserRgy	Determines which type of user registry is configured for the Security Access Manager policy server.
PDPolicy constructor	Instantiates a policy object for a user, or for all users in the case of the global policy.
PDPolicy object.acctDisableTimeEnforced	Returns an indicator whether the account disable time interval policy is enforced.
PDPolicy object.acctDisableTimeUnlimited	Returns an indicator whether the account disable time interval policy is unlimited.
PDPolicy object.acctExpDateEnforced	Returns an indicator whether the account expiration date policy is enforced.
PDPolicy object.acctExpDateUnlimited	Returns an indicator whether the account expiration date policy is unlimited.
PDPolicy object.getAcctExpDate	Returns the account expiration date for user accounts.
PDPolicy object.getAcctDisableTimeInterval	Returns the amount of time to disable a user account when the maximum number of login failures is exceeded.
PDPolicy object.PDPolicy.getMaxConcurrentWebSessions	Returns the maximum concurrent web sessions that are allowed.
PDPolicy object.getMaxFailedLogins	Returns the maximum number of failed logins that are allowed for user accounts.
PDPolicy object.getAccessibleDays *PDPolicy object*.getAccessStartTime *PDPolicy object*.getAccessEndTime *PDPolicy object*.getAccessTimezone	Returns the time of day access policy for user accounts.
PDPolicy.maxConcurrentWebSessionsDisplaced	Returns an indicator whether the maximum concurrent web sessions policy is displaced.
PDPolicy.maxConcurrentWebSessionsEnforced	Returns an indicator whether the maximum concurrent web sessions policy is enforced.
PDPolicy.maxConcurrentWebSessionsUnlimited	Returns an indicator whether the maximum concurrent web sessions policy is unlimited.

Table 5. Administering user account policies (continued)

Method	Description
PDPolicy object.maxFailedLoginsEnforced	Returns an indicator whether the maximum failed login policy is enforced.
PDPolicy.setAcctExpDate *PDPolicy object*.setAcctExpDate	Sets the account expiration date for user accounts.
PDPolicy.setAcctDisableTime *PDPolicy object*.setAcctDisableTime	Sets the amount of time to disable a user account when the maximum number of login failure is exceeded.
PDPolicy.setMaxConcurrentWebSessions *PDPolicy object*.PDPolicy.setMaxConcurrentWebSessions	Sets the maximum concurrent Web sessions that are allowed.
PDPolicy.setMaxFailedLogins *PDPolicy object*.setMaxFailedLogins	Sets the maximum number of failed logins allowed for user accounts.
PDPolicy.setTodAccess *PDPolicy object*.setTodAccess	Sets the time of day access for the account for user accounts. **Note:** When setting a password policy, you provide a list of days, start time, and end time. The start time and end time apply to each day on the list. If the specified start time is later than the specified end time, then the access is allowed until the specified end time is reached the next day.
PDPolicy object.todAccessEnforced	Returns an indicator whether the time-of-day access policy is enforced.

For detailed reference information about these methods, see the Javadoc HTML documentation. For information about accessing this documentation, see Table 1 on page 3.

Administering user password policies

You can manage user access by setting password attributes. You can specify policies that apply either only to a single user or for all users.

When a user password policy attribute is set to a value and enforced, the value always takes precedence over the value that is set for the general policy. This is true regardless of which value is more restrictive. If a password policy attribute for a user is not enforced, then the value is set for the general policy, if that value is set and enforced, is in effect for the user.

Table 6 on page 28 describes the administration API methods that you can use to modify or access password policies.

Table 6. Administering user password policies

Method	Description
PDPolicy constructor	Instantiates a policy object for a user, or for all users in the case of the global policy.
PDPolicy object.getMaxPwdAge	Returns the password expiration date.
PDPolicy object.getMaxPwdRepChars	Returns the maximum number of repeated characters that are allowed in the password.
PDPolicy object.getMinPwdAlphas	Returns the minimum number of alphabetic characters that are allowed in the password.
PDPolicy object.getMinPwdLen	Returns the minimum password length.
PDPolicy object.getMinPwdNonAlphas	Returns the minimum number of nonalphabetic characters that are allowed in a password.
PDPolicy object.maxPwdAgeEnforced	Returns an indicator whether the maximum password age policy is enforced.
PDPolicy object.maxPwdRepCharsEnforced	Returns an indicator whether the password maximum repeated characters policy is enforced.
PDPolicy object.minPwdAlphasEnforced	Returns an indicator whether the password minimum alphabetic characters required policy is enforced.
PDPolicy object.minPwdLenEnforced	Returns an indicator whether the minimum password length policy is enforced.
PDPolicy object.minPwdNonAlphasEnforced	Returns an indicator whether the password minimum non-alphabetic characters policy is enforced.
PDPolicy object.pwdSpacesAllowed	Returns an indicator whether spaces are allowed in a password.
PDPolicy.setMaxPwdAge *PDPolicy object*.setMaxPwdAge	Sets the password expiration date.
PDPolicy.setMaxPwdRepChars *PDPolicy object*.setMaxPwdRepChars	Sets the maximum number of repeated characters that are allowed in a password.
PDPolicy.setMinPwdAlphas *PDPolicy object*.setMinPwdAlphas	Sets the minimum number of alphabetic characters that are allowed in a password.
PDPolicy.setMinPwdLen *PDPolicy object*.setMinPwdLen	Sets the minimum password length.
PDPolicy.setMinPwdNonAlphas *PDPolicy object*.setMinPwdNonAlphas	Sets the minimum number of non-alphabetic characters that are allowed in a password.
PDPolicy.setPwdSpacesAllowed *PDPolicy object*.setPwdSpacesAllowed	Sets policy for whether spaces are allowed in a password.

For detailed reference information about these methods, see the Javadoc HTML documentation. For information about accessing this documentation, see Table 1 on page 3.

Administering groups

The administration API provides methods for creating, accessing, listing, and deleting Security Access Manager group information from the user registry.

The name of a group is not case-sensitive. The following group names all pertain to the same Security Access Manager group:

- `group`
- `GROUP`
- `Group`
- `GrOuP`

The `PDGroup.createGroup` method creates a group in the user registry that is used by the Security Access Manager policy server.

Note: When a group definition exists in the user registry, use the `PDGroup.importGroup` method instead.

The `PDGroup.importGroup` method imports an existing group definition from the user registry into Security Access Manager and allows the group definition to be managed by Security Access Manager.

Table 7 lists the group administration functions.

Table 7. Administering groups

Method	Description
PDGroup.createGroup	Creates the specified group.
PDGroup.importGroup	Creates a Security Access Manager group by importing an existing group from the user registry.
PDGroup.deleteGroup	Deletes the specified group.
PDGroup.listGroups	Lists Security Access Manager groups.

For detailed reference information about these methods, see the Javadoc HTML documentation. For information about accessing this documentation, see Table 1 on page 3.

Administering group information

You can manage information that is associated with a group by using the administration API.

When a group is created in the user registry, you can set and get different pieces of information about the group. You must create a security context between the calling application and the Security Access Manager policy server before you can access the user registry. You can obtain the user registry information for a group object by specifying either the Security Access Manager group name or the user registry group name.

Table 8 lists the group information administration functions.

Table 8. Administering group attributes

Method	Description
PDGroup constructor	Instantiates a group object for the specified Security Access Manager or user registry name.
PDGroup object.getDescription	Returns the group description.
PDGroup object.getRgyName	Returns the user registry name for the group.
PDGroup object.getId	Returns the Security Access Manager name for the group.
PDGroup object.isPDGroup	Returns an indicator whether the object is a Security Access Manager group.
PDGroup.setDescription *PDGroup object*.setDescription	Sets a group description.
PDGroup object.getMembers	Lists the members of a group.
PDGroup.addMembers *PDGroup object*.addMembers	Adds users to a group.
PDGroup.removeMembers *PDGroup object*.removeMembers	Removes users from a group.

For detailed reference information about these methods, see the Javadoc HTML documentation. For information about accessing this documentation, see Table 1 on page 3.

Chapter 4. Administering protected objects and protected object spaces

You can use the administration API to create, modify, examine, list, and delete Security Access Manager protected objects.

These protected objects represent resources that must be secured to enforce your security policy. You can specify the security policy by applying access control lists (ACLs), protected object policies (POPs), and authorization rules to the protected objects.

Security Access Manager protected objects exist within a virtual hierarchy known as a *protected object space*. Security Access Manager provides several protected object spaces by default. You can use the administration API to define new regions of the protected object space and to define and secure resources that are specific to a third-party application.

This chapter describes the administration API functions that you can use to administer protected object spaces and protected objects.

You must be familiar with protected objects before using the administration API. For an introduction to protected objects, see the chapter about managing protected objects in the *IBM Security Access Manager for Web: Administration Guide*.

For an introduction to the use of ACLs, POPs, and authorization rules to secure protected objects, see the chapters about using access control policies, protected object policies, and authorization rules in the *IBM Security Access Manager for Web: Administration Guide*.

This chapter contains the following topics:

Administering protected object spaces

You can use the administration API to create and administer a user-defined protected object space.

You can use this protected object space to define a resource hierarchy that is specific to a third-party application that uses Security Access Manager authorization services to enforce a security policy.

User-defined object spaces that are created with the administration API are dynamic because they can be updated while Security Access Manager is running.

Table 9 on page 32 lists the methods available for administering protected object spaces.

Note: For an introduction to the creation of protected object spaces, see the protected object space information in the *IBM Security Access Manager for Web: Administration Guide*.

Table 9. Administering protected object spaces

Methods	Description
PDProtObjectSpace.createProtObjectSpace	Creates a Security Access Manager protected object space.
PDProtObjectSpace.deleteProtObjectSpace	Deletes the specified Security Access Manager protected object space.
PDProtObjectSpace.listProtObjectSpaces	Lists the Security Access Manager protected object spaces.

For detailed reference information about these methods, see the Javadoc HTML documentation. For information about accessing this documentation, see Table 1 on page 3.

Administering protected objects

Define protected objects that reflect the resources that your security policy protects.

The name of a protected object can be of any length and contain any character. The forward slash (/) character is interpreted to be part of the object hierarchy, which allows ACLs to be attached at the various points that are indicated by the forward slash character.

After you create a protected object, you can specify a security policy for it by defining and attaching ACLs, POPs, authorization rules, or any combination of these entities.

For more information about the Security Access Manager security concepts, see the *IBM Security Access Manager for Web: Administration Guide*.

When you implement protected objects programmatically, use caution. In many cases, the protected object hierarchy is manually designed, built, and tested by a security expert. Carefully review the hierarchy to ensure that the security policy is correctly enforced. If you choose to build protected object hierarchies programmatically, be sure to test and review the settings for each object before you deploy the security environment.

Table 10 lists the methods available to administer protected objects.

Table 10. Administering protected objects

Methods	Description
PDProtObject.attachAcl *PDProtObject object*.attachACL	Attaches the specified access control list to the specified protected object.
PDProtObject.attachPop *PDProtObject object*.attachPop	Attaches a POP to the specified protected object.
PDProtObject.attachAuthzRule *PDProtObj object*.attachAuthzRule	Attaches an authorization rule to the specified protected object.
PDProtObject.createProtObject	Creates a Security Access Manager protected object.
PDProtObject.deleteProtObject	Deletes the specified Security Access Manager protected object.

Table 10. Administering protected objects (continued)

Methods	Description
PDProtObject.detachAcl *PDProtObject object*.detachAcl	Detaches the access control list from the specified protected object.
PDProtObject.detachPop *PDProtObject object*.detachPop	Detaches a POP from the specified protected object.
PDProtObject.detachAuthzRule *PDProtObj object*.detachAuthzRule	Detaches an authorization rule from the specified protected object.
PDProtObject constructor	Instantiates the specified protected object. If the protected object name specified does not exist, default values are shown. To determine that a protected object exists, use `PDProtObject.exists`.
PDProtObject object.getAclId	Gets the name of the ACL attached to the specified protected object.
PDProtObject object.getEffectiveAclId	Gets the name of the ACL in effect for the specified protected object.
PDProtObject object.getPopId	Gets the name of the POP attached to the specified protected object.
PDProtObject object.getEffectivePopId	Gets the name of the POP in effect for the specified protected object.
PDProtObj object.getAuthzRuleId	Gets the name of the authorization rule object that is attached to the specified protected object.
PDProtObj object.getEffectiveAuthzRuleId	Gets the name of the authorization rule object that is in effect for the specified protected object.
PDProtObject object.getDescription	Gets the description of the specified protected object.
PDProtObject object.getId	Gets the name of the specified protected object.
PDProtObject object.isPolicyAttachable	Indicates whether a protected object policy or access control list can be attached to the specified protected object.
PDProtObject object.exists	Indicates whether a protected object exists.
PDProtObject object.access	Indicates whether a specific action to a specific object is permitted.
PDProtObject object.multiAccess	Indicates whether the specified actions to the specified objects are permitted.
PDProtObject.listProtObjectsByPop	Returns a list of protected objects that have the specified protected object policy (POP) attached.
PDProtObject.listProtObjects	Returns the protected objects contained under the specified directory.
PDProtObject.listProtObjectsByAcl	Returns a list of protected objects that have the specified access control list attached.
PDProtObject.setDescription *PDProtObject object*.setDescription	Sets the description field of the specified protected object.
PDProtObject.setPolicyAttachable *PDProtObject object*.setPolicyAttachable	Sets whether a protected object policy or access control list can be attached to the specified protected object.

Table 10. Administering protected objects (continued)

Methods	Description
PDProtObj.listProtObjectsByAuthzRule	Lists the protected objects that have the specified authorization rule attached.

For detailed reference information about these methods, see the Javadoc HTML documentation. For information about accessing this documentation, see Table 1 on page 3.

Administering extended attributes for a protected object

The extended attributes for a protected object can be created, set, queried, and deleted.

Protected objects without explicitly defined extended attributes inherit the first found set of extended attributes, which are defined at the parent object within the inheritance chain. The found set of extended attributes replaces the empty set of defined attributes. These inherited attributes are called *effective extended attributes*.

Table 11 describes the methods for administering extended attributes and effective extended attributes for a protected object.

Table 11. Administering protected object attributes

Methods	Description
PDProtObject.deleteAttribute *PDProtObject object*.deleteAttribute	Deletes the specified extended attribute (name and values) from the specified protected object.
PDProtObject.deleteAttributeValue *PDProtObject object*.deleteAttributeValue	Deletes the specified value from the specified extended attribute key in the specified protected object.
PDProtObject.getEffectiveAttributeValues	Displays a list of the values for the effective extended attribute that is associated with the specified protected object.
PDProtObject.getEffectiveAttributeNames	Displays a list of all the effective extended attributes that are associated with the specified protected object.
PDProtObject.getEffectiveAttributeObjectId	Displays the name of the protected object that has the extended attributes defined. When no extended attributes are defined, NULL is returned.
PDProtObject object.getAttributeValues	Returns the values that are associated with the specified extended attribute for the specified protected object.
PDProtObject object.getAttributeNames	Lists all the extended attributes that are associated with the specified protected object.
PDProtObject.setAttributeValue *PDProtObject object*.setAttributeValue	Creates an extended attribute with the specified name and value, if it does not exist, and adds the attribute to the specified protected object. If the attribute specified exists, the specified value is added to the existing attribute.

For detailed reference information about these methods, see the Javadoc HTML documentation. For information about accessing this documentation, see Table 1 on page 3.

Chapter 5. Administering access control

You can use the administration API to create, modify, examine, list, and delete Security Access Manager access control lists (ACLs).

Use the administration API to attach ACLs to Security Access Manager protected objects, and to detach ACLs from protected objects.

Each ACL might contain entries for specific users and groups. You can use the administration API to set ACL entries for users and groups that exist in the Security Access Manager secure domain. You can also use the administration API to set ACL entries for the default user categories any-other and unauthenticated.

ACL entries consist of one or more permissions. These permissions specify actions that the owner of the entry is allowed to perform. Security Access Manager provides a number of default permissions. You can use the administration API to define additional extended actions. You also can use the administration API to group the extended actions into action groups.

Understand the construction and use of ACLs before using the administration API ACL functions. The proper use of ACLs is key to successfully implementing a security policy. For more information, see the chapter about using access control lists in the *IBM Security Access Manager for Web: Administration Guide*.

This chapter contains the following topics:

Administering access control lists

You can allow or restrict specific users and groups from accessing protected resources by using access control lists (ACLs).

You can do the following tasks with the administration API:

- Create and delete ACLs
- Retrieve or change information that is associated with an ACL
- List the user, group, any-other, and unauthenticated entries that are included in the ACL
- List all defined ACLs

The name of an ACL can be of any length. The following characters are allowed in an ACL name:

- Alphanumeric characters that are defined in the locale
- The underscore (_) character
- The hyphen (-) character

You can specify the following items:

- User entries that belong in each ACL
- Permissions or actions that each user is allowed to perform
- Permissions or actions that are based on group membership, rather than individual user identity, to expedite administration tasks

The administration API defines the `PDAcl` object to contain a retrieved ACL. You can use administration API , classes, and methods to extract information from the `ivadmin_aclPDAcl` object.

Be sure that you understand how to define an ACL policy before you use the administration API ACL methods . For more information, see the section about ACL entry syntax in the *IBM Security Access Manager for Web: Administration Guide*.

Table 12 describes the methods for administering ACLs.

Table 12. Administering access control lists

Methods	Description
PDAcl.createAcl	Creates new ACL.
PDAcl.deleteAcl	Deletes the specified ACL.
PDAcl constructor	Instantiates the specified ACL.
PDAcl object.getDescription	Returns the description of the specified ACL.
PDAcl object.getId	Returns the name of the specified ACL.
PDAcl.listAcls	Returns the names of all the defined ACLs.
PDAcl.setDescription *PDAcl object*.setDescription	Sets or modifies the description for the specified ACL.

For detailed reference information about these methods, see the Javadoc HTML documentation. For information about accessing this documentation, see Table 1 on page 3.

Administering access control list entries

You must create an ACL object before you can administer ACL entries for the object.

The administration API can be used to specify entries for each of the following ACL entry types:

- Users
- Groups
- User any-other (also known as any-authenticated)
- User unauthenticated

PDAclEntryUser
An ACL entry that applies to a particular user.

PDAclEntryGroup
An ACL entry that applies to all members of a particular group.

PDAclEntryAnyOther
The ACL entry that applies to any other authenticated users. Any user that is already authenticated into the Security Access Manager secure domain,

but is not covered by a separate user or group entry in the access control list, is allowed the permissions that are specified by this ACL entry.

PDAclEntryUnAuth
The ACL entry that applies to unauthenticated users. Any user that is not already authenticated is allowed the permissions that are specified by this ACL entry.

Be sure that you understand ACL entry syntax, ACL entry types, and ACL permission (action) attributes before you use the administration API methods in this section.

Security Access Manager supports 18 default actions. For a list of the default Security Access Manager actions, see the section about default Security Access Manager permissions for actions in the *IBM Security Access Manager for Web: Administration Guide*.

For more information, see the section about ACL entry syntax in the *IBM Security Access Manager for Web: Administration Guide*.

Table 13 lists the methods for administering ACL entries.

Table 13. Administering access control list entries

Methods	Description
PDAcl object.getPDAclEntryAnyOther	Returns the `PDAclEntryAnyOther` object that is associated with the ACL.
PDAcl object.getPDAclEntryUnAuth	Returns the `PDAclEntryUnAuth` object that is associated with the ACL.
PDAcl object.getPDAclEntriesUser	Returns a Java `HashMap` of the `PDAclEntryUser` objects that are associated with the ACL.
PDAcl object.getPDAclEntriesGroup	Returns a Java `HashMap` of the `PDAclEntryGroup` objects that are associated with the ACL.
PDAcl.removePDAclEntryAnyOther *PDAcl object*.removePDAclEntryAnyOther	Removes the ACL entry for the `any-other` user from the specified ACL.
PDAcl.removePDAclEntryGroup *PDAcl object*.removePDAclEntryGroup	Removes the ACL entry for the specified group from the specified ACL.
PDAcl.removePDAclEntryUnAuth *PDAcl object*.removePDAclEntryUnAuth	Removes the ACL entry for the `unauthenticated` user from the specified ACL.
PDAcl.removePDAclEntryUser *PDAcl object*.removePDAclEntryUser	Removes the ACL entry for the specified user from the specified ACL.
PDAcl.setPDAclEntryAnyOther *PDAcl object*.setPDAclEntryAnyOther	Sets or modifies the ACL entry for the `any-other` user in the ACL. Call this function to specify permissions for all authenticated users who do not have a separate user or group entry in the specified ACL.
PDAcl.setPDAclEntryGroup *PDAcl object*.setPDAclEntryGroup	Sets or modifies the ACL entry for the specified group in the specified ACL.

Table 13. Administering access control list entries (continued)

Methods	Description
PDAcl.setPDAclEntryUnAuth *PDAcl object*.setPDAclEntryUnAuth	Sets the ACL entry for the `unauthenticated` user in the specified ACL. Call this function to specify permissions for those users that are not already authenticated.
PDAcl.setPDAclEntryUser *PDAcl object*.setPDAclEntryUser	Sets the entry for the specified user in the specified ACL. Use this function to specify the actions that a user is permitted to perform.

For detailed reference information about these methods, see the Javadoc HTML documentation. For information about accessing this documentation, see Table 1 on page 3.

Administering access control list extended attributes

Extended attributes for an ACL can be obtained, set, and deleted.

Table 14 lists the methods available for administering ACL extended attributes.

Table 14. Administering access control list extended attributes

Methods	Description
PDAcl.deleteAttribute *PDAcl object*.deleteAttribute	Deletes the specified extended attribute key from the specified ACL.
PDAcl.deleteAttributeValue *PDAcl object*.deleteAttributeValue	Deletes the specified value from the specified extended attribute key in the specified ACL.
PDAcl object.getAttributeValues	Gets the extended attribute values for the specified extended attribute key from the specified ACL.
PDAcl object.getAttributeNames	Lists the extended attribute keys associated with the specified ACL.
PDAcl.setAttributeValue *PDAcl object*.setAttributeValue	Creates an extended attribute with the specified name and value, if it does not exist, and adds the attribute to the specified ACL. If the attribute specified exists, the specified value is added to the existing values for the attribute.

For detailed reference information about these methods, see the Javadoc HTML documentation. For information about accessing this documentation, see Table 1 on page 3.

Administering action groups

You can use the administration API to create, examine, and delete new action groups.

Each action group can contain up to 32 actions. The default action group, referred to as the primary action group, contains the 18 predefined Security Access Manager actions, which means you can create up to 14 new actions to the primary group.

When you need to create more than 32 actions, you can use the administration API to define a new action group. Security Access Manager supports up to 32 action groups.

For more information about action groups, see the section about creating extended ACL actions and action groups in the *IBM Security Access Manager for Web: Administration Guide*. Table 15 lists the methods for administering action groups.

Table 15. Administering action groups

Methods	Description
PDActionGroup.createActionGroup	Creates new action group with the specified name.
PDActionGroup.deleteActionGroup	Deletes the specified action group and all the actions that belong to the specified group.
PDActionGroup.listActionGroups	Lists all the defined action group names.

For detailed reference information about these methods, see the Javadoc HTML documentation. For information about accessing this documentation, see Table 1 on page 3.

Administering extended actions

Security Access Manager provides a default set of actions (permissions) that belong to the primary action group that can be granted to users or groups. You can use the administration API to define new, extended actions that supplement the set of default actions. Each of the extended actions can belong to the primary action group or to a custom action group.

Extended actions are typically defined to support actions that are specific to a third-party application. For more information about extended actions, see the section about creating extended ACL actions and action groups in the *IBM Security Access Manager for Web: Administration Guide*.

Table 16 lists the methods for administering extended actions.

Table 16. Administering extended actions

Methods	Description
PDAction.createAction	Defines a new action (permission) in the specified action group.
PDAction.deleteAction	Deletes an action (permission) from the specified action group.
PDAction constructor	Gets the specified `PDAction` object.
PDAction object.getDescription	Returns the description for the specified action.
PDAction object.getId	Returns the name for the specified action.

Table 16. Administering extended actions (continued)

Methods	Description
PDAction object.getType	Returns the type for the specified action.
PDAction.listActions	Lists all the defined actions (permissions) for the specified action group.

For detailed reference information about these methods, see the Javadoc HTML documentation. For information about accessing this documentation, see Table 1 on page 3.

Chapter 6. Administering protected object policies

Use the administration API to create, modify, examine, and delete Security Access Manager protected object policies (POPs).

You can also use the Administration API to attach or detach POPs from protected objects.

You can use POPs to impose more conditions on operations that are included in the access control list (ACL) policy. These additional conditions are enforced regardless of the user or group identities that are specified in the ACL entries.

See the following examples of the conditions:

- Specifying the quality of protection
- Writing a report record to the auditing service
- Requiring an authentication strength level
- Restricting access to a specific time period
- Enabling or disabling the warning mode, which allows an administrator to validate security policy

You must understand the Security Access Manager POP concepts before you use the administration API to administer POPs. For more information, see the chapter about using POPs in the *IBM Security Access Manager for Web: Administration Guide*.

See the details in the following topics:

- "Administering protected object policy objects"
- "Administering protected object policy settings" on page 45
- "Administering protected object policy extended attributes" on page 46

Administering protected object policy objects

POP objects are administered in a similar way to ACL policies. You can create and configure a POP, and then attach the POP to objects in the protected object space.

Table 17 lists the methods for administering protected object policy objects.

Table 17. Administering protected object policy objects

Method	Description
PDPop.createPop	Creates a POP object with the default values.
PDPop.deletePop	Deletes the specified POP.
PDPop object.getDescription	Returns the description of the specified POP.
PDPop object.getId	Returns the name of the specified POP.
PDProtObject.listProtObjectsByPop	Finds and lists all protected objects that have the specified POP attached.
PDPop constructor *PDProtObject object*.getPop	Returns the specified POP object.
PDPop.listPops	Lists all POP objects.

For detailed reference information about these methods, see the Javadoc HTML documentation. For information about accessing this documentation, see Table 1 on page 3.

PDPop.IPAuthInfo object

An array of `PDPop.IPAuthInfo` objects is passed as input to the `PDPop.setIPAuthInfo` and `PDPop.removeIPAuthInfo` methods.

Each `PDPop.IPAuthInfo` object contains the following information:

- The IP address that is associated with the credentials that are being checked.
- The netmask that is associated with the credentials that are being checked.
- The IP authentication level of the credentials for the specified IP address and netmask that are used when accessing the protected object to which this POP is attached. All integer values except `1000` are supported for specifying a level index. Use the constant `IPAUTH_LEVEL_FORBIDDEN_ALL_NETWORKS` to deny access from all networks.

The IP address and netmask can be specified in either of the following formats:

IPv4 The primary format of an IPv4 IP address is `x.x.x.x`, which is a 32-bit numeric address that is written as four numbers that are separated by periods. A value of `0.0.0.0` indicates that this setting is for any other network for which this policy is not set explicitly.

IPv6

One of the primary formats of an IPv6 IP address is `x:x:x:x:x:x:x:x`, which is a 128-bit numeric address that is written as eight numbers that are separated by colons. The contiguous fields that contain only the digits zero can be collapsed (for example: `0009:0000:0000:0000:0000:0008:0007:0006` can be represented as `9::8:7:6`).

A zero network and netmask value indicates that this setting is for any other network for which this policy is not set explicitly. See the standard RFC 2373 to determine what constitutes a valid representation of an IPv6 address. Security Access Manager does not support prefix notation.

Note: When you specify the IP address or netmask, be aware of the following restrictions:

- IPv4 clients must provide addresses in IPv4 format to IPv4 servers.
- IPv4 clients can provide addresses in IPv4 or IPv6 format to IPv6 servers.

For an IPv6 address to be accepted, the server must be IPv6. You cannot provide an IPv6 address to an IPv4 server.

See the *IBM Security Access Manager for Web: Administration Guide* for more information about the IP authentication POP policy. See the Javadoc information for the `PDPop.IPAuthInfo` object and its associated methods for more information.

Administering protected object policy settings

You can use the administration API to set, modify, or remove attributes in a POP.

You must create the POP object before you specify POP settings.

You can use administration API functions to specify the following POP attributes:

- Authentication levels
- Quality of Protection (QOP) requirements
- Auditing levels
- Time of day access restrictions
- Warning mode settings

Authentication levels specify whether more or alternative authentication is required to access a protected object. The additional authentication is also called step-up authentication. This means that an additional authentication step is required to access resources that require more restrictive access policies. When you use step-up authentication, you can filter users according to their IP addresses, or you can specify step-up authentication for all users, regardless of IP address.

For more information about the use of the authentication level by WebSEAL, see the section about authentication strength POP policy (step-up) in the *IBM Security Access Manager for Web: Web Security Developer Reference*.

The quality of protection (QOP) level is not enforced internally by Security Access Manager. Applications that set the quality of protection can enforce it.

Audit levels specify what operations generate an audit record. This value is used internally by Security Access Manager and also can be used by applications to generate their audit records.

The time of day access setting is used to control access to a protected object based on the time when the access occurs.

Note: When you modify a protected object policy, you must provide a list of days, start time, and end time. The start time and end time apply to each day on the list. If the specified start time is greater than the specified end time, then the access is allowed until the specified end time of the next day.

The warning mode enables a security administrator to troubleshoot the authorization policy set on the protected object space.

When you set the warning attribute to `yes`, any action is possible by any user on the object where the POP is attached. Any object can be accessed even if the ACL policy attached to the object is set to deny this access.

Audit records are generated that capture the results of all ACL policies with warning mode set throughout the object space. The audit log shows the outcome of an authorization decision as it is made if the warning attribute is set to `no`.

Table 18 on page 46 lists the methods for administering protected object policy settings.

Table 18. Administering protected object policy settings

Methods	Description
PDPop object.getIPAuthInfo	Returns the IP authentication level information from the specified POP.
PDPop object.getAuditLevel	Returns the audit level for the specified POP.
PDPop object.getQOP	Returns the quality of protection (QOP) level for the specified POP.
PDPop object.getTodAccessInfo	Returns the time of day range for the specified POP.
PDPop object.getWarningMode	Returns the warning mode value from the specified POP.
PDPop.removeIPAuthInfo *PDPop object*.removeIPAuthInfo	Removes the specified IP authentication level information from the specified POP.
PDPop.setIPAuthInfo *PDPop object*.setIPAuthInfo	Sets the IP authentication level information for the specified POP.
PDPop.setAuditLevel *PDPop object*.setAuditLevel	Sets the audit level for the specified POP.
PDPop.setDescription *PDPop object*.setDescription	Sets the description of the specified POP.
PDPop.setQOP *PDPop object*.setQOP	Sets the quality of protection level for the specified POP.
PDPop.setTodAccessInfo *PDPop object*.setTodAccessInfo	Sets the time of day range for the specified POP.
PDPop.setWarningMode *PDPop object*.setWarningMode	Sets the warning mode for the specified POP.

For detailed reference information about these methods, see the Javadoc HTML documentation. For information about accessing this documentation, see Table 1 on page 3.

Administering protected object policy extended attributes

Table 19 lists the methods for administering protected object policy extended attributes.

Table 19. Administering protected object policy extended attributes

Methods	Description
PDPop.deleteAttribute *PDPop object*.deleteAttribute	Deletes the specified extended attribute from the specified POP.
PDPop.deleteAttributeValue *PDPop object*.deleteAttributeValue	Deletes the specified value from the specified extended attribute key in the specified POP.

Table 19. Administering protected object policy extended attributes (continued)

Methods	Description
PDPop object.getAttributeValues	Gets the values for the specified extended attribute from the specified POP.
PDPop object.getAttributeNames	Lists the extended attributes associated with the specified POP.
PDPop.setAttributeValue *PDPop object*.setAttributeValue	Sets the value for the specified extended attribute in the specified POP.

For detailed reference information about these methods, see the Javadoc HTML documentation. For information about accessing this documentation, see Table 1 on page 3.

Chapter 7. Administering authorization rules

Authorization rules are conditions or standards that are contained in an authorization policy that are used to make access decisions that are based on attributes such as user, application, and environment context. Authorization rules are defined to specify conditions that must be met before access to a protected object is permitted. A rule is created by using Boolean conditions that are based on data that is supplied to the authorization engine within the user credential, from the resource manager application or from the encompassing business environment.

A Security Access Manager authorization rule is a policy type like an access control list (ACL) or a protected object policy (POP). The rule is stored as a text rule within a rule policy object and is attached to a protected object in the same way and with the same constraints as ACLs and POPs.

The Security Access Manager administration Java classes provide methods to create, delete, modify, list, and get authorization rules.

For more information about authorization rules, see the *IBM Security Access Manager for Web: Administration Guide*.

Use the methods shown in Table 20 to administer authorization rule objects.

Table 20. Administering authorization rules

Method	Description
PDAuthzRule.createAuthzRule	Creates the specified authorization rule object.
PDAuthzRule.deleteAuthzRule	Deletes the specified authorization rule object.
PDAuthzRule constructor	Instantiates the specified authorization rule object.
PDAuthzRule object.getId	Returns the ID for the specified authorization rule.
PDAuthzRule object.getDescription	Returns the description for the specified authorization rule.
PDAuthzRule object.getFailReason	Returns the fail reason, if any, for the specified authorization rule.
PDAuthzRule object.getRuleText	Returns the rule text for the specified authorization rule.
PDAuthzRule.listAuthzRules	Lists all the registered authorization rules.
PDAuthzRule.setDescription *PDAuthzRule object*.setDescription	Sets the description for the specified authorization rule.
PDAuthzRule.setRuleText *PDAuthzRule object*.setRuleText	Sets the authorization rule text.
PDAuthzRule.setFailReason *PDAuthzRule object*.setFailReason	Sets the authorization rule fail reason.

For detailed reference information about these methods, see the Javadoc HTML documentation. For information about accessing this documentation, see Table 1 on page 3.

Chapter 8. Administering single sign-on resources

You can use the administration API to administer resources that enable a Security Access Manager user to obtain single sign-on (SSO) capability across more than one web server.

This capability requires the use of Security Access Manager WebSEAL junctions.

You can use the administration API to create, modify, examine, and delete the following types of resources:

- Administering web resources
- Administering resource groups
- Administering resource credentials

Be sure that you understand Security Access Manager single sign-on support before you use the administration API to administer single sign-on resources.

For more information about administering single sign-on capability across junctioned web server resources, see the section about user registry resource management commands in the *IBM Security Access Manager for Web: Administration Guide* and the section about using global sign-on (GSO) in the *IBM Security Access Manager for Web: Web Security Developer Reference*.

This chapter contains the following topics:

Administering Web resources

A *Web resource* is a Web server that serves as the backend of a Security Access Manager WebSEAL junction. An application on the joined Web server can require users to authenticate specifically to the application.

The authentication information, such as user name and password, often differs from the authentication information used by Security Access Manager. Because of this difference, the junctioned Web server requires an authenticated Security Access Manager user to log in again, using the user name and password specific to the application on the joined Web server.

You can use the administration API to configure Security Access Manager so that Security Access Manager users must authenticate only one time. You must define a Web resource (server) and then define a user-specific resource credential that contains user-specific authentication information for the Web resource.

This section describes how to create, modify, and delete Web resources. Administration of resource credentials is described in "Administering resource credentials" on page 53.

Note: The administration API does not perform all WebSEAL junction configuration tasks through the API. Use the `pdadmin` commands to modify the junction definitions. For more information, see the *IBM Security Access Manager for Web: WebSEAL Administration Guide*.

Table 21 lists the methods for administering Web resources.

Table 21. Administering Web resources

Methods	Description
PDSSOResource.createSSOResource	Creates a single sign-on Web resource.
PDSSOResource.deleteSSOResource	Deletes the specified single sign-on Web resource.
PDSSOResource constructor	Instantiates the specified single sign-on Web resource.
PDSSOResource object.getDescription	Returns the description of the specified single sign-on Web resource.
PDSSOResource object.getId	Returns the name (identifier) of the specified single sign-on Web resource.
PDSSOResource.listSSOResources	Returns a list of all the single sign-on Web resource names.

For detailed reference information about these methods, see the Javadoc HTML documentation. For information about accessing this documentation, see Table 1 on page 3.

Administering resource groups

A *resource group* is a group of web servers that are connected to a Security Access Manager WebSEAL server and that use the same set of user IDs and passwords.

You can use the administration API to create resource groups. You can then create a single resource credential for all the resources in the resource group. A single resource credential lets you simplify the management of web resources by grouping similar web resources into resource groups.

You can also use the administration API to add more web resources, when necessary, to an existing resource group.

Table 22 lists the methods for administering resource groups.

Table 22. Administering resource groups

Methods	Description
PDSSOResourceGroup.addSSOResource *PDSSOResourceGroup object*.addSSOResource	Adds a single sign-on resource to a single sign-on resource group.
PDSSOResourceGroup.createSSOResourceGroup	Creates a single sign-on group resource.
PDSSOResourceGroup.deleteSSOResourceGroup	Deletes a single sign-on group resource.
PDSSOResourceGroup constructor	the specified single sign-on group resource.
PDSSOResourceGroup object.getDescription	Returns the description of the single sign-on group resource.

Table 22. Administering resource groups (continued)

Methods	Description
PDSSOResourceGroup object.getId	Returns the name of the single sign-on group resource.
PDSSOResourceGroup object.getSSOResources	Returns a list of the member single sign-on resource names for the specified single sign-on group.
PDSSOResourceGroup.listSSOResourceGroups	Returns a list of all single sign-on group resource names.
PDSSOResourceGroup.removeSSOResource *PDSSOResourceGroup object*.removeSSOResource	Removes a single sign-on resource from the specified single sign-on resource group.

Depending on the LDAP server in your environment, any attempt to remove a non-existing resource from a group, might generate an error.

For detailed reference information about these methods, see the Javadoc HTML documentation. For information about accessing this documentation, see Table 1 on page 3.

Administering resource credentials

A *resource credential* provides a user ID and password for a single sign-on user-specific resource, such as a Web server or a group of Web servers. The Web resource or group of Web resources must exist before you can apply resource credentials to it.

Resource credential information is stored in the Security Access Manager entry in the user registry.

You can use the administration API to create, modify, examine, and delete resource credentials.

Table 23 lists the methods for administering credentials.

Table 23. Administering credentials

Methods	Description
PDSSOCred.createSSOCred	Creates a single sign-on credential.
PDSSOCred.deleteSSOCred	Deletes a single sign-on credential.
PDSSOCred constructor	the specified single sign-on credential.
PDSSOCred object.getResourceName	Returns the name of the single sign-on resource associated with this credential.
PDSSOCred object.getResourcePassword	Returns the password associated with this single sign-on credential.
PDSSOCred object.getResourceUser	Returns the name of the resource user associated with the specified single sign-on credential.
PDSSOCred object.getResourceType	Returns the type of the single sign-on resource associated with the specified single sign-on credential.

Table 23. Administering credentials (continued)

Methods	Description
PDSSOCred object.getUser	Returns the name of the Security Access Manager user associated with this single sign-on credential.
PDSSOCred.listAndShowSSOCreds	Returns the list of single sign-on credentials for the specified user.
PDSSOCred.listSSOCreds	Returns the IDs (user, resource, and type) of the single sign-on credentials for the specified user. This information is a subset of that returned by the `listAndShowSSOCreds` method.
PDSSOCred.setSSOCred *PDSSOCred object*.setSSOCred	Modifies a single sign-on credential.

For detailed reference information about these methods, see the Javadoc HTML documentation. For information about accessing this documentation, see Table 1 on page 3.

Chapter 9. Administering domains

A Security Access Manager domain consists of all the physical resources that require protection along with the associated security policy used to protect those resources.

The initial domain is the management domain and is created when the Policy Server is configured. Multiple domains can exist simultaneously within a Security Access Manager environment.

Data is securely partitioned between domains. A user or process must authenticate to a specific domain to access data contained within it.

Each Security Access Manager environment contains a single management domain. A user must be authenticated to the management domain to create, delete, list, or modify additional domains.

To specify the management domain in methods that take a domain argument, use the **PDDomain.getMgmtDomainName** method.

Each Java Runtime Environment (JRE) can optionally be configured to use a specific domain. This domain is called the local domain. To specify the local domain in methods that take a domain argument, use the `PDDomain.getLocalDomainName` method. If a JRE is not configured to use a specific domain, the local domain defaults to the management domain.

The Java classes provide methods that can be used to manage domains.

For more information about the management of domains, see the *IBM Security Access Manager for Web: Administration Guide*. Table 24 lists the methods for administering domains.

Table 24. Administering domains

Methods	Description
PDDomain.createDomain	Creates new Security Access Manager domain.
PDDomain.deleteDomain	Deletes the specified Security Access Manager domain.
PDDomain constructor	Instantiates the specified domain object.
PDDomain object.getDescription	Gets the description for the specified Security Access Manager domain.
PDDomain object.getId	Gets the name of the specified Security Access Manager domain.
PDDomain.listDomains	Lists the names of all the Security Access Manager domains, except for the management domain.
PDDomain.getLocalDomainName	Gets the name of the local domain.
PDDomain.getMgmtDomainName	Gets the name of the management domain.
PDDomain.setDescription *PDDomain object*.setDescription	Changes the description for the specified Security Access Manager domain.

For detailed reference information about these methods, see the Javadoc HTML documentation. For information about accessing this documentation, see Table 1 on page 3.

Chapter 10. Configuring application servers

You can use the administration API to configure and unconfigure authorization and administration servers, modify configuration parameters, administer replicas, and perform certificate maintenance.

The `com.tivoli.pd.jcfg.SvrSslCfg class` is used to perform the necessary configuration steps that allow an application to use a secure sockets layer (SSL) connection for communicating with the policy server or the authorization server.

It is not intended to do all the configuration that might be required to ensure a correctly functioning application. For more information about the `com.tivoli.pd.jcfg.SvrSslCfg` class, see the *IBM Security Access Manager for Web: Authorization Java Classes Developer Reference*.

This chapter contains the following topics:

Configuring application servers

Use the configuration commands to enable an application server (an application that uses the authorization or administration API) to communicate with the policy server or the authorization server. An administrative user identity (for example, **sec_master**) and password must be specified for connecting to the policy server.

Table 25. Configuring application servers

Methods	Description
PDAppSvrConfig.configureAppSvr	Configures an application server by updating the configuration file and creating the `keystore` file.
PDAppSvrConfig.setAppSvrListening	Sets or resets the enable-listening parameter in the configuration file.
PDAppSvrConfig.setAppSvrDbDir	Sets the local policy database directory in the configuration file.
PDAppSvrConfig.setAppSvrDbRefresh	Sets the local policy database refresh interval in the configuration file
PDAppSvrConfig.setAppSvrPort	Changes the listening port number of the application in the configuration file.
PDAppSvrConfig.unconfigureAppSvr	Unconfigures an application server.

For detailed reference information about these methods, see the Javadoc HTML documentation. For information about accessing this documentation, see Table 1 on page 3.

Administering configuration information

Use the configuration commands to add, change, or delete replica entries in the configuration file as well as return other configuration information.

Table 26. Administering configuration information

Methods	Description
PDAppSvrConfig.addPDServer	Adds a replica entry to the configuration file.
PDAppSvrConfig.changePDServer	Changes parameters of a replica entry in the configuration file.
PDAppSvrConfig.removePDServer	Removes a replica entry from the configuration file.
PDAppSvrConfig.getPDAppSvrInfo	Returns a `PDAppSvrInfo` object containing information stored in the configuration file.
PDAppSvrConfig.getKeystoreURL	Returns the URL of the keystore file that is associated with the configuration file.

For detailed reference information about these methods, see the Javadoc HTML documentation. For information about accessing this documentation, see Table 1 on page 3.

Certificate maintenance

Only use `ivadmin_cfg_renewservercert()` the `replaceAppSvrCert` method when the certificate has been compromised.

Table 27. Certificate maintenance

Methods	Description
PDAppSvrConfig.replaceAppSvrCert	and replaces the server SSL certificate.

For detailed reference information about these methods, see the Javadoc HTML documentation. For information about accessing this documentation, see Table 1 on page 3.

Chapter 11. Administering servers

You can use the administration API to get a list of tasks from the server, send a specific task to an authorization server, and notify replica databases, either automatically or manually, when the master authorization database is updated.

This chapter contains the following topics:

Getting and performing administration tasks

You can send an administration task to a server. You also can request a list of all supported administration tasks from a server. The caller must have credentials with sufficient permission to perform the task. For more information, see the *IBM Security Access Manager for Web: Authorization C API Developer Reference*.

Notifying replica databases when the master authorization database is updated

When an administrator makes security policy changes, the policy server adjusts to the master authorization database to reflect these changes.

To ensure that these changes also are dispersed to any authorization servers with replica databases, you can do one or more of the following:

- Configure a Security Access Manager application server, such as WebSEAL, to poll the master authorization database at regular intervals for updates. By default, polling is disabled. For more information about polling the master authorization database, see the `cache-refresh-interval` option described in the *IBM Security Access Manager for Web: Authorization C API Developer Reference*.
- Enable the policy server to notify authorization servers each time that the master authorization database is updated. This automatic process is recommended for environments where database changes are infrequent. For more information, see "Notifying replica databases automatically" on page 60.
- Notify authorization servers, on demand, after you make updates to the master authorization database. This manual process is recommended for environments where database changes are frequent and involve substantial changes. For instructions, see "Notifying replica databases manually" on page 60.

After you select the method that you want to use to update replica databases (automatic, manual, or both), you can fine-tune settings in the `ivmgrd.conf` file on the policy server. For more information, see "Setting the maximum number of notification threads" on page 60 and "Setting the notification wait time" on page 60.

Notifying replica databases automatically

You can enable the policy server to send notifications to authorization servers each time that the master authorization database is updated. In turn, the authorization servers automatically request a database update from the policy server.

To enable automatic database updates, edit the `ivmgrd.conf` file on the policy server and add the following *attribute=value* stanza entry pair:

```
[ivmgrd]
auto-database-update-notify = yes
```

Restart the policy server for changes to take effect. Use the setting for environments where the master database is not changed frequently. To turn off automatic notification, specify no.

Notifying replica databases manually

When the master authorization database is updated, you can use the `PDServer.replicateServer` method to send notifications to application servers that are configured to receive database update notifications.

You can indicate that a specific server receive update notifications, or specify NULL, which notifies all configured authorization servers in the secure domain.

If you specify a server name, you are notified whether the server was replicated successfully or if a failure occurred. If you do not specify a server name, return codes indicate whether the policy server started notifying authorization servers in your secure domain.

Unless you specify the *server-name* option, you are not notified when an authorization server database was replicated successfully.

Setting the maximum number of notification threads

When the master authorization database is updated, this update is announced to replica databases through the use of *notification threads*. Each replica then has the responsibility of downloading the new data from the master authorization database.

You can edit the `ivmgrd.conf` file to set a value for the maximum number of notification threads. This number is calculated based on the number of replica databases in your secure domain. For example, if you have 10 replica databases and want to notify them of master database changes simultaneously, specify a value of 10 for the `max-notifier-threads` stanza entry as shown:

```
[ivmgrd]
max-notifier-threads = 10
```

The default value is 10 threads.

Setting the notification wait time

There is a time delay between when the policy server updates the master authorization database and when notification is sent to database replicas.

If you added the `auto-database-update-notify = yes` stanza entry to the `ivmgrd.conf` file as described in "Notifying replica databases automatically," you can set this period.

To do so, edit the `notifier-wait-time` stanza entry value in the `ivmgrd.conf` file. For example, if you are making batch changes to the master authorization database, wait until all changes are finished before policy changes are sent to database replicas. As a result, you might decide to increase the default value from 15 seconds to 25 seconds as shown:

```
[ivmgrd]
notifier-wait-time = 25
```

By editing the value for this attribute, the policy server is prevented from sending individual replica notifications for each of a series of database changes.

Administering servers and database notification

Table 28. Administering servers and database notification

Methods	Description
PDServer *constructor*	Instantiates a server object.
PDServer object.getAdminServices	Returns the list of Administration Services registered by this server.
PDServer object.getDescription	Returns the description of this server.
PDServer object.getHostName	Returns the host name of this server.
PDServer object.getId	Returns the identifier of this server.
PDServer object.getPort	Returns the port of this server.
PDServer object.getTaskList	Returns a list of tasks from the server.
PDServer object.getUserId	Returns the user identifier of this server.
PDServer.listServers	Lists all the registered servers.
PDServer.performTask	Sends a command to an authorization server.
PDServer.replicateServer	Notifies authorization servers to receive database updates.

For detailed reference information about these methods, see the Javadoc HTML documentation. For information about accessing this documentation, see Table 1 on page 3.

Appendix A. Differences between the C and Java administration API

If you are familiar with the administration C API, you must be aware of several notable differences between them and the administration Java classes and methods that are described in this document.

In particular, the handling of security context management and response processing are different between the two implementations. In addition, there are other subtle differences outlined in this appendix.

For more information about the administration C API, see *IBM Security Access Manager for Web: Administration C API Developer Reference*.

Security context management differences

The `ivadmin_context_create3()` function in the C language administration API creates a communication connection to the Security Access Manager policy server.

The context object that is returned by this function is tightly coupled to an actual Secure Sockets Layer (SSL) session. When the SSL session times out, the user must delete the context and create a new one to reestablish communication with the policy server.

You must delete the contexts which are not needed on a timely basis with `ivadmin_context_delete()` to free SSL resources. As a result, the programmer has the responsibility to manage SSL sessions by using context objects and the `ivadmin_context_*` APIs.

The Java implementation of the context, which uses the `PDContext` object, hides the management of the actual SSL sessions from the user. The `PDContext` object contains only the information that is needed to establish communication with the server: the server location, authentication information of the client, and the locale to be used for message translation.

The `PDContext` objects are not tied to a particular SSL session. Instead, an SSL session is obtained when a `PDContext` object is used in a Java method invocation. Security Access Manager manages all aspects of the SSL sessions itself, including creating them, pooling them, reusing them, and eventually deleting them, without any explicit context management from the programmer.

Response processing differences

Most of the C language administration API functions return a value that indicates the overall success or failure of the requested operation. They also return an `ivadmin_response` object as an output parameter. This response object contains optional messages that can be after processed using the `ivadmin_response_*` functions.

The Java language administration API methods throw a `PDException` exception on failure. Most methods provide a `PDMessages` output as an output parameter. This

object contains optional messages that can be after processed using the accessor methods provided in the `PDMessages` object class.

Additional differences

The following additional differences exist between the C language and Java language implementations of the Security Access Manager administration API:

- The method names in the `PDUser` and `PDGroup` classes are user-registry neutral. The function names provided in the administration C APIs are specific to Lightweight Directory Access Protocol (LDAP) specific. This difference arises from the continuing support of a wider range of user registries in Security Access Manager.
- The user and group names that appear in the methods associated with the `PDUser` and `PDGroup` classes are structured to allow for the possible future addition of other user registries.
- The type field is not supported in the `PDProtObject` and `PDProtObjectSpace` classes. Use extended attributes to provide equivalent function. This difference arises from the confusion caused by the type field on the administration C APIs not being used internally in earlier Security Access Manager versions.
- The administration Java classes and methods provide both certificate-based and user ID and password-based authentication. The administration C API provides only user ID and password-based authentication.
- The `svrsslcfg` command-line interface (CLI) can be used only for applications written using the administration C API. For Java applications, use the `com.tivoli.pd.jcfg.SvrSslCfg` Java class instead.
- Policy information, such as maximum password age, is encapsulated in a `PDPolicy` class instead of being defined in the user and context objects as it is in the administration C API. The function provided is the same whether using the Java classes or the C API.
- When using the administration C APIs, the user must renegotiate the security context when a session time-out occurs. The `PDContext` class handles this processing automatically.

Appendix B. Deprecated Java classes and methods

For information about the deprecated Java classes and methods, see the Javadoc HTML documentation.

For details about accessing this HTML documentation, see Table 1 on page 3.

Existing Java applications must be changed to use the indicated replacement class or method.

Appendix C. Administration API equivalents

This appendix shows the mapping that exists between the administration C API functions, the administration Java classes and methods, the `pdadmin` commands, and Web Portal Manager.

In some cases, an operation can be performed in different ways. In some cases two or more method calls might be necessary to achieve the same effect as a single C API function.

For more information, see the following books.

- Information about the administration C API can be found in the *IBM Security Access Manager for Web: Administration C API Developer Reference*.
- Additional information about the administration Java classes and methods can be found in the Javadoc information.
- Information about the `pdadmin` commands can be found in the *IBM Security Access Manager for Web: Command Reference*.
- Information about Web Portal Manager can be found in its online help and in the *IBM Security Access Manager for Web: Administration Guide*.

Table 29. Mapping of the administration C API to the Java methods, the pdadmin interface, and Web Portal Manager

C API	Java class and method	pdadmin command equivalent	Web Portal Manager equivalent
`ivadmin_acl_attrdelkey()`	**PDAcl.deleteAttribute** *PDAcl object*.**deleteAttribute**	`pdadmin acl modify` *`acl_name`* `delete attribute` *`attribute_name`*	**ACL** → **List ACL** → click ACL name → **Extended Attribute** tab → select attributes → **Delete**
`ivadmin_acl_attrdelval()`	**PDAcl.deleteAttributeValue** *PDAcl object*.**deleteAttributeValue**	`pdadmin acl modify` *`acl_name`* `delete attribute` *`attribute_name attribute_value`*	Not supported
`ivadmin_acl_attrget()`	*PDAcl object*.**getAttributeValues**	`pdadmin acl show` *`acl_name`* `attribute` *`attribute_name`*	**ACL** → **List ACL** → click ACL name → **Extended Attribute** tab
`ivadmin_acl_attrlist()`	*PDAcl object*.**getAttributeNames**	`pdadmin acl list` *`acl_name`* `attribute`	**ACL** → **List ACL** → click ACL name → **Extended Attribute** tab
`ivadmin_acl_attrput()`	**PDAcl.setAttributeValue** *PDAcl object*.**setAttributeValue**	`pdadmin acl modify` *`acl_name`* `set attribute` *`attribute_name attribute_value`*	**ACL** → **List ACL** → click ACL name → **Extended Attribute** tab → **Create** → fill in form → **Apply**
`ivadmin_acl_create()`	**PDAcl.createAcl**	`pdadmin acl create` *`acl_name`*	**ACL** → **Create ACL** → fill in form → **Create**
`ivadmin_acl_delete()`	**PDAcl.deleteAcl**	`pdadmin acl delete` *`acl_name`*	**ACL** → **List ACL** → select ACL names → **Delete**
`ivadmin_acl_get()`	**PDAcl** constructor	`pdadmin acl show` *`acl_name`*	**ACL** → **List ACL** → click ACL name
`ivadmin_acl_getanyother()`	*PDAcl object*.**getPDAclEntryAnyOther**	`pdadmin acl show any-other`	**ACL** → **List ACL** → click **Any-other**
`ivadmin_acl_ getdescription()`	*PDAcl object*.**getDescription**	`pdadmin acl show` *`acl_name`*	**ACL** → **List ACL** → click ACL name
`ivadmin_acl_getgroup()`	*PDAcl object*.**getPDAclEntriesGroup**	`pdadmin acl show` *`acl_name`*	**ACL** → **List ACL** → click ACL name
`ivadmin_acl_getid()`	*PDAcl object*.**getId**	`pdadmin acl show` *`acl_name`*	**ACL** → **List ACL** → click ACL name
`ivadmin_acl_getunauth()`	*PDAcl object*.**getPDAclEntryUnAuth**	`pdadmin acl show` *`acl_name`*	**ACL** → **List ACL** → click ACL name

Table 29. Mapping of the administration C API to the Java methods, the pdadmin interface, and Web Portal Manager (continued)

C API	Java class and method	pdadmin command equivalent	Web Portal Manager equivalent
ivadmin_acl_getuser()	*PDAcl object*.**getPDAclEntriesUser**	pdadmin acl show *acl_name*	**ACL** → **List ACL** → click ACL name
ivadmin_acl_list()	**PDAcl.listAcls**	pdadmin acl list	**ACL** → **List ACL**
ivadmin_acl_listgroups()	*PDAcl object*.**getPDAclEntriesGroup**	pdadmin acl show *acl_name*	**ACL** → **List ACL** → click ACL name
ivadmin_acl_listusers()	*PDAcl object*.**getPDAclEntriesUser**	pdadmin acl show *acl_name*	**ACL** → **List ACL** → click ACL name
ivadmin_acl_removeanyother()	**PDAcl.removePDAclEntryAnyOther** *PDAcl object*.**removePDAclEntry AnyOther**	pdadmin acl modify *acl_name* remove any-other	**ACL** → **List ACL** → click ACL name → select **Any-other** → **Delete**
ivadmin_acl_removegroup()	**PDAcl.removePDAclEntryGroup** *PDAcl object*.**removePDAclEntryGroup**	pdadmin acl modify *acl_name* remove group *group_name*	**ACL** → **List ACL** → click ACL name → select group name → **Delete**
ivadmin_acl_removeunauth()	**PDAcl.removePDAclEntryUnAuth** *PDAcl object*.**removePDAclEntry UnAuth**	pdadmin acl modify *acl_name* remove unauthenticated	**ACL** → **List ACL** → click ACL name → select **Unauthenticated** → **Delete**
ivadmin_acl_removeuser()	**PDAcl.removePDAclEntryUser** *PDAcl object*.**removePDAclEntryUser**	pdadmin acl modify *acl_name* remove user *user_name*	**ACL** → **List ACL** → click ACL name → select user name → **Delete**
ivadmin_acl_setanyother()	**PDAcl.setPDAclEntryAnyOther** *PDAcl object*.**setPDAclEntryAnyOther**	pdadmin acl modify *acl_name* set any-other *permissions*	**ACL** → **List ACL** → click ACL name → select **Any-other** → **Create** → select permissions → **Apply**
ivadmin_acl_ setdescription()	**PDAcl.setDescription** *PDAcl object*.**setDescription**	pdadmin acl modify *acl_name* description *description*	**ACL** → **List ACL** → click ACL name → modify description → **Set**
ivadmin_acl_setgroup()	**PDAcl.setPDAclEntryGroup** *PDAcl object*.**setPDAclEntryGroup**	pdadmin acl modify *acl_name* set group *group_name permissions*	**ACL** → **List ACL** → click ACL name → **Create** → select **Group**→ specify group name → select permissions → **Apply**
ivadmin_acl_setunauth()	**PDAcl.setPDAclEntryUnAuth** *PDAcl object*.**setPDAclEntryUnAuth**	pdadmin acl modify *acl_name* set unauthenticated *permissions*	**ACL** → **List ACL** → click ACL name → **Create** → select **Unauthenticated** → select permissions → **Apply**
ivadmin_acl_setuser()	**PDAcl.setPDAclEntryUser** *PDAcl object*.**setPDAclEntryUser**	pdadmin acl modify *acl_name* set user *user_name permissions*	**ACL** → **List ACL** → click ACL name → **Create** → select **User** → specify user name → select permissions → **Apply**
ivadmin_action_create()	**PDAction.createAction**	pdadmin action create *name description action_type*	**ACL** → **List Action Groups** → click primary action group → **Create** → fill in form → **Create**
ivadmin_action_create_in_group()	**PDAction.createAction**	pdadmin action create *name description action_type action_group_name*	**ACL** → **List Action Groups** → click action group → **Create** → fill in form → **Create**
ivadmin_action_delete()	**PDAction.deleteAction**	pdadmin action delete *name*	**ACL** → **List Action Groups** → select primary action group → select actions → **Delete**
ivadmin_action_delete_from _group()	**PDAction.deleteAction**	pdadmin action delete *name action_group_name*	**ACL** → **List Action Groups** → select action group → select actions → **Delete**
ivadmin_action_getdescription()	*PDAction object*.**getDescription**	pdadmin action list	**ACL** → **List Action Groups** → click primary action group
ivadmin_action_getid()	*PDAction object*.**getId**	pdadmin action list	**ACL** → **List Action Groups** → click primary action group
ivadmin_protobj_gettype()	*PDAction object*.**getType**	pdadmin action list	**ACL** → **List Action Groups** → click primary action group
ivadmin_action_gettype()	**PDActionGroup.createActionGroup**	pdadmin action group create *action_group_name*	**ACL** → **Create Action Group** → type group name → **Create**
ivadmin_action_group_create()	**PDActionGroup.deleteActionGroup**	pdadmin action group delete *action_group_name*	**ACL** → **List Action Groups** → select action groups → **Delete**

Table 29. Mapping of the administration C API to the Java methods, the pdadmin interface, and Web Portal Manager (continued)

C API	Java class and method	pdadmin command equivalent	Web Portal Manager equivalent
ivadmin_action_group_delete()	**PDActionGroup.listActionGroups**	pdadmin action group list	**ACL** → **List Action Groups**
ivadmin_action_list()	**PDAction.listActions**	pdadmin action list	**ACL** → **List Action Groups** → click primary action group
ivadmin_action_list_in_group()	**PDAction.listActions**	pdadmin action list *action_group_name*	**ACL** → **List Action Groups** → click action group
ivadmin_authzrule_create()	**PDAuthzRule.createAuthzRule**	pdadmin authzrule create *ruleid* –rulefile {*filename* | *ruletext*} [–desc *description*] [–failreason *failreason*]	**AuthzRule** → **Create AuthzRule** → fill in form → **Create**
ivadmin_authzrule_delete()	**PDAuthzRule.deleteAuthzRule**	pdadmin authzrule delete *ruleid*	**AuthzRule** → **List AuthzRule** → select authorization rule name → **Delete**
ivadmin_authzrule_get()	**PDAuthzRule** constructor	pdadmin authzrule show *ruleid*	**AuthzRule** → **List AuthzRule** → click authorization rule name
ivadmin_authzrule_getdescription()	*PDAuthzRule object*.**getDescription**	pdadmin authzrule show *ruleid*	**AuthzRule** → **List AuthzRule** → click authorization rule name
ivadmin_authzrule_ getfailreason()	*PDAuthzRule object*.**getFailReason**	pdadmin authzrule show *ruleid*	**AuthzRule** → **List AuthzRule** → click authorization rule name
ivadmin_authzrule_getid()	*PDAuthzRule object*.**getId**	pdadmin authzrule show *ruleid*	**AuthzRule** → **List AuthzRule** → click authorization rule name
ivadmin_authzrule_getruletext()	*PDAuthzRule object*.**getRuleText**	pdadmin authzrule show *ruleid*	**AuthzRule** → **List AuthzRule** → click authorization rule name
ivadmin_authzrule_list()	**PDAuthzRule.listAuthzRules**	pdadmin authzrule list	**AuthzRule** → **List AuthzRule**
ivadmin_authzrule_ setdescription()	**PDAuthzRule.setDescription** *PDAuthzRule object*.**setDescription**	pdadmin authzrule modify *ruleid* description *description*	**AuthzRule** → **List AuthzRule** → click authorization rule name → **General** tab → modify fields → **Apply**
ivadmin_authzrule_ setfailreason()	**PDAuthzRule.setFailReason** *PDAuthzRule object*.**setFailReason**	pdadmin authzrule modify *ruleid* failreason *failreason*	**AuthzRule** → **List AuthzRule** → click authorization rule name → **General** tab → modify fields → **Apply**
ivadmin_authzrule_setruletext()	**PDAuthzRule.setRuleText** *PDAuthzRule object*.**setRuleText**	pdadmin authzrule modify *ruleid* –rulefile {*filename* | *ruletext*}	**AuthzRule** → **List AuthzRule** → click authorization rule name → modify fields → **Apply**
ivadmin_cfg_addreplica2()	**PDAppSvrConfig.addPDServer**	svrsslcfg -add_replica -f *cfg_file* -h *host_name* [-p *port*] [-k *rank*]	Not supported.
ivadmin_cfg_chgreplica2()	**PDAppSvrConfig.changePDServer**	svrsslcfg -chg_replica -f *cfg_file* -h *host_name* [-p *port*] [-k *rank*]	Not supported.
ivadmin_cfg_configureserver3()	**PDAppSvrConfig.configureAppSvr**	svrsslcfg -config -f *cfg_file* -d *kdb_dir_name* -n *server_name* ...	Not supported.
ivadmin_cfg_getvalue()	Not supported.	pdadmin config show *config_file stanza key*	Not supported.
ivadmin_cfg_removevalue()	Not supported.	pdadmin config modify keyvalue remove *config_file stanza key* [*value*]	Not supported.
ivadmin_cfg_renewservercert()	**PDAppSvrConfig.replaceAppSvrCert**	svrsslcfg -chgcert -f *cfg_file* -n *server_name* [-A *admin_ID*] -P *admin_pwd*	Not supported.
ivadmin_cfg_rmvreplica2()	**PDAppSvrConfig.removePDServer**	svrsslcfg -rmv_replica -f *cfg_file* -h *host_name* [-p *port*] [-k *rank*]	Not supported.

Table 29. Mapping of the administration C API to the Java methods, the pdadmin interface, and Web Portal Manager (continued)

C API	Java class and method	pdadmin command equivalent	Web Portal Manager equivalent
ivadmin_cfg_ setapplicationcert2()	Not supported.	svrsslcfg -modify -f *cfg_file* [-t *timeout*] [-C *cert_file*] [-l *listening_mode*]	Not supported.
ivadmin_cfg_setkeyringpwd2()	Not applicable.	svrsslcfg -chgpwd -f *cfg_file* -n *server_name* [-A *admin_ID*] [-P *admin_pwd*]	Not supported.
ivadmin_cfg_setlistening2()	**PDAppSvrConfig.setAppSvrListening**	svrsslcfg -f *cfg_file* -modify -l yes	Not supported.
ivadmin_cfg_setport2()	**PDAppSvrConfig.setAppSvrPort**	svrsslcfg -config -f *cfg_file* -d *kdb_dir_name* -n *server_name* ...	Not supported.
ivadmin_cfg_setssltimeout2()	Not supported.	svrsslcfg -modify -f *cfg_file* -t *timeout* [-C *cert_file*] [-l *listening_mode*]	Not supported.
ivadmin_cfg_setsvrpwd()	Not supported.	pdadmin config modify svrpassword *config_file password*	Not supported.
ivadmin_cfg_setvalue()	Not supported.	pdadmin config modify keyvalue { set | append }[–obfuscate] *config_file stanza key* [*value*]	Not supported.
ivadmin_cfg_unconfigureserver()	**PDAppSvrConfig.unconfigureAppSvr**	svrsslcfg -unconfig -f *cfg_file* -n *server_name* [-A *admin_ID*] -P *admin_pwd*	Not supported.
ivadmin_context_cleardelcred()	*PDContext object*.**clearDelegatedCred**	Not applicable.	Not applicable.
ivadmin_context_create3()	**PDContext** constructor	Not applicable.	Not applicable.
ivadmin_context_createdefault2()	**PDContext** constructor	Not applicable.	Not applicable.
ivadmin_context_createlocal()	Not supported.	Not applicable.	Not applicable.
ivadmin_context_delete()	*PDContext object*.**close**	Not applicable.	Not applicable.
ivadmin_context_ domainismanagement()	*PDContext object*.**domainIsManagement**	pdadmin context show	Not supported.
ivadmin_context_getaccexpdate()	*PDPolicy object*.**getAcctExpDate**	pdadmin policy get account-expiry-date	**User → Show Global User Policy** → view Account Expiration Date section
ivadmin_context_getcodeset()	*PDContext object*.**getLocale**	Not applicable.	Not applicable.
ivadmin_context_getdisabletimeint()	*PDPolicy object*.***getAcctDisableTimeInterval***	pdadmin policy get disable-time-interval	**User → Show Global User Policy** → view Disable Time Interval section
ivadmin_context_getdomainid()	*PDContext object*.**getDomainid**	pdadmin context show	Not supported.
Not supported.	Not supported.	pdadmin errtext *error_number*	Not supported.
Not supported.	Not supported.	pdadmin exit	Not supported.
ivadmin_context_ getmaxconcurwebsess()	**PDPolicy.getMaxconcurrentWebSessions** *PDPolicy object* **.getMaxconcurrentWebSessions**	pdadmin policy get max-concurrent-web-sessions	**User → Show Global User Policy** → view Max Concurrent Web Sessions section
ivadmin_context_getmaxlgnfails()	*PDPolicy object*.**getMaxFailedLogins**	pdadmin policy get max-login-failures	**User → Show Global User Policy** → view Max Login Failures section
ivadmin_context_getmaxpwdage()	*PDPolicy object*.**getMaxPwdAge**	pdadmin policy get max-password-age	**User → Show Global User Policy** → view Max Password Age section
ivadmin_context_ getmaxpwdrepchars()	*PDPolicy object*.**getMaxPwdRepChars**	pdadmin policy get max-password-repeated-chars	**User → Show Global User Policy** → view Max Password Repeated Characters section
ivadmin_context_ getmgmtdomainid()	**PDDomain.getMgmtDomainName**	pdadmin login –m	Initial login.
ivadmin_context_getmgmtsvrhost()	Not supported.	Not supported.	Not available.

Table 29. Mapping of the administration C API to the Java methods, the pdadmin interface, and Web Portal Manager (continued)

C API	Java class and method	pdadmin command equivalent	Web Portal Manager equivalent
ivadmin_context_getmgmtsvrport()	Not supported.	Not supported.	Not available.
ivadmin_context_getminpwdalphas()	*PDPolicy object*.**getMinPwdAlphas**	pdadmin policy get min-password-alphas	**User** → **Show Global User Policy** → view Minimum Password Alphas section
ivadmin_context_getminpwdlen()	*PDPolicy object*.**getMinPwdLen**	pdadmin policy get min-password-length	**User** → **Show Global User Policy** → view Minimum Password Length section
ivadmin_context_ getminpwdnonalphas()	*PDPolicy object*.**getMinPwdNonAlphas**	pdadmin policy get min-password-non-alphas	**User** → **Show Global User Policy** → view Minimum Password Non-Alphas section
ivadmin_context_getpwdspaces()	*PDPolicy object*.**pwdSpacesAllowed**	pdadmin policy get password-spaces	**User** → **Show Global User Policy** → view Password Spaces Allowed section
ivadmin_context_gettodaccess()	*PDPolicy object*.**getAccessibleDays** *PDPolicy object*.**getAccessStartTime** *PDPolicy object*.**getAccessEndTime** *PDPolicy object*.**getAccessTimezone**	pdadmin policy get tod-access	**User** → **Show Global User Policy** → view Time of Day Access section
ivadmin_context_getuserid()	*PDContext object*.**getUserid**	pdadmin context show	Not supported.
ivadmin_context_getuserreg()	**PDUser.getUserRgy**	pdadmin admin show configuration	Not supported.
ivadmin_context_hasdelcred()	*PDContext object*.**hasDelegatedCred**	Not applicable.	Not applicable.
ivadmin_context_setaccexpdate()	**PDPolicy.setAcctExpDate** *PDPolicy object*.**setAcctExpDate**	pdadmin policy set account-expiry-date {unlimited | *absolute_time* | unset}	**User** → **Show Global User Policy** → set Account Expiration Date → **Apply**
ivadmin_context_setdelcred()	*PDContext object*.**setDelegatedCred**	Not applicable.	Not applicable.
ivadmin_context_ setdisabletimeint()	**PDPolicy.setAcctDisableTime** *PDPolicy object*.**setAcctDisableTime**	pdadmin policy set disable-time-interval {*number* | unset | disable}	**User** → **Show Global User Policy** → set Account Disable Time Interval → **Apply**
ivadmin_context_ setmaxconcurwebsess()	**PDPolicy.setMaxconcurrent WebSessions** *PDPolicy object* **.setMaxconcurrent WebSessions**	pdadmin policy set max-concurrent-web-sessions {*number*| displace | unlimited | unset} -user *user_name*	**User** → **Show Global User Policy** → set Max Concurrent Web Sessions → **Apply**
ivadmin_context_setmaxlgnfails()	**PDPolicy.setMaxFailedLogins** *PDPolicy object*.**setMaxFailedLogins**	pdadmin policy set max-login-failures {*number* | unset}	**User** → **Show Global User Policy** → set Max Login Failures → **Apply**
ivadmin_context_setmaxpwdage()	**PDPolicy.setMaxPwdAge** *PDPolicy object*.**setMaxPwdAge**	pdadmin policy set max-password-age {*relative_time* | unset}	**User** → **Show Global User Policy** → set Max Password Age → **Apply**
ivadmin_context_setmaxpwdrepchars()	**PDPolicy.setMaxPwdRepChars** *PDPolicy object*.**setMaxPwdRepChars**	pdadmin policy set max-password-repeated-chars [*number* | unset]	**User** → **Show Global User Policy** → set Max Password Repeated Characters → **Apply**
ivadmin_context_ setminpwdalphas()	**PDPolicy.setMinPwdAlphas** *PDPolicy object*.**setMinPwdAlphas**	pdadmin policy set min-password-alphas {*number* | unset}	**User** → **Show Global User Policy** → set Minimum Password Alphas → **Apply**
ivadmin_context_ setminpwdlen()	**PDPolicy.setMinPwdLen** *PDPolicy object*.**setMinPwdLen**	pdadmin policy set min-password-length {*number* | unset}	**User** → **Show Global User Policy** → set Minimum Password Length → **Apply**
ivadmin_context_setminpwd nonalphas()	**PDPolicy.setMinPwdNonAlphas** *PDPolicy object*.**setMinPwdNonAlphas**	pdadmin policy set max-password-non-alphas {*number* | unset}	**User** → **Show Global User Policy** → set Minimum Password Non-Alphas → **Apply**
ivadmin_context_setpwdspaces()	**PDPolicy.setPwdSpacesAllowed** *PDPolicy object*.**setPwdSpacesAllowed**	pdadmin policy set password-spaces {yes | no | unset}	**User** → **Show Global User Policy** → set Password Spaces Allowed → **Apply**
ivadmin_context_settodaccess()	**PDPolicy.setTodAccess** *PDPolicy object*.**setTodAccess**	pdadmin policy set tod-access *todaccess_value*	**User** → **Show Global User Policy** → set Time of Day Access → **Apply**

Table 29. Mapping of the administration C API to the Java methods, the pdadmin interface, and Web Portal Manager (continued)

C API	Java class and method	pdadmin command equivalent	Web Portal Manager equivalent
`ivadmin_domain_create()`	**PDDomain.createDomain**	`pdadmin domain create` *domain domain_admin_id domain_admin_password* `[–desc` *description*`]`	**Secure Domain** → **Create Secure Domain** → fill in form → **Create**
`ivadmin_domain_delete()`	**PDDomain.deleteDomain**	`pdadmin domain delete` *domain* `[–registry]`	**Secure Domain** → **List Secure Domain** → select secure domain names → **Delete**
`ivadmin_domain_get()`	**PDDomain** constructor	`pdadmin domain show` *domain*	**Secure Domain** → **List Secure Domain** → click secure domain name
`ivadmin_domain_getdescription()`	*PDDomain object*.**getDescription**	`pdadmin domain show` *domain*	**Secure Domain** → **List Secure Domain** → click secure domain name
`ivadmin_domain_getid()`	*PDDomain object*.**getId**	`pdadmin domain show` *domain*	**Secure Domain** → **List Secure Domain** → click secure domain name
`ivadmin_domain_list()`	**PDDomain.listDomains**	`pdadmin domain list`	**Secure Domain** → **List Secure Domain**
`ivadmin_domain_setdescription()`	**PDDomain.setDescription** *PDDomain object*.**setDescription**	`pdadmin domain modify` *domain* `description` *description*	**Secure Domain** → **List Secure Domain** → click secure domain name → modify description → **Apply**
`ivadmin_free()`	Not applicable.	Not applicable.	Not applicable.
`ivadmin_group_addmembers()`	**PDGroup.addMembers** *PDGroup object*.**addMembers**	`pdadmin group modify` *group_name* `add` *user* `pdadmin group modify` *group_name* `add (`*user_1 user_2* `[ ...` *user_n*`])`	**Group** → **Search Groups** → type pattern and maximum results → **Search** → click group name → **Members** tab → select users → **Add**
`ivadmin_group_create2()`	**PDGroup.createGroup**	`pdadmin group create` *group_name dn cn* `[`*group_container*`]`	**Group** → **Create Group** → fill in form → **Create**
`ivadmin_group_delete2()`	**PDGroup.deleteGroup**	`pdadmin group delete [-registry]` *group_name*	**Group** → **Search Groups** → type pattern and maximum results → **Search** → select group names → **Delete**
`ivadmin_group_get()`	**PDGroup** constructor	`pdadmin group show` *group_name*	**Group** → **Search Groups** → type pattern and maximum results → **Search** → click group name
`ivadmin_group_getbydn()`	**PDGroup** constructor	`pdadmin group show-dn` *dn*	Not supported.
`ivadmin_group_getcn()`	Not supported.	`pdadmin group show` *group_name*	**Group** → **Search Groups** → type pattern and maximum results → **Search** → click group name
`ivadmin_group_getdescription()`	*PDGroup object*.**getDescription**	`pdadmin group show` *group_name*	**Group** → **Search Groups** → type pattern and maximum results → **Search** → click group name
`ivadmin_group_getdn()`	*PDGroup object*.**getRgyName**	`pdadmin group show` *group_name*	**Group** → **Search Groups** → type pattern and maximum results → **Search** → click group name
`ivadmin_group_getid()`	*PDGroup object*.**getId**	`pdadmin group show` *group_name*	**Group** → **Search Groups** → type pattern and maximum results → **Search** → click group name
`ivadmin_group_getmembers()`	*PDGroup object*.**getMembers**	`pdadmin group show-members` *group_name*	**Group** → **Search Groups** → type pattern and maximum results → **Search** → click group name → **Members** tab

Table 29. Mapping of the administration C API to the Java methods, the pdadmin interface, and Web Portal Manager (continued)

C API	Java class and method	pdadmin command equivalent	Web Portal Manager equivalent
`ivadmin_group_import2()`	*PDGroup object*.**importGroup**	`pdadmin group import` *group_name dn* [*group_container*]	**Group** → **Import Group** → fill in form → **Import**
`ivadmin_group_list()`	**PDGroup.listGroups**	`pdadmin group list` *pattern max_return*	**Group** → **Search Groups** → type pattern and maximum results → **Search**
`ivadmin_group_listbydn()`	**PDGroup.listGroups**	`pdadmin group list-dn` *pattern max_return*	Not supported.
`ivadmin_group_removemembers()`	**PDGroup.removeMembers** *PDGroup object*.**removeMembers**	`pdadmin group modify` *group_name* `remove` *user* `pdadmin group modify` *group_name* `remove (`*user_1 user_2* `[ ... ` *user_n*`])`	**Group** → **Search Groups** → type pattern and maximum results → **Search** → click group name → **Members** tab → select user names → **Remove**
`ivadmin_group_setdescription()`	**PDGroup.setDescription** *PDGroup object*.**setDescription**	`pdadmin group modify` *group_name* `description` *description*	**Group** → **Search Groups** → type pattern and maximum results → **Search** → click group name → type description → **Apply**
Not supported	Not supported	`pdadmin help {`*topic* | *command*`}`	Not supported
Not supported	Not supported	`pdadmin login –a admin_id –p` *password* `[–d` *domain* | `–m]`	Not supported
Not supported	Not supported	`pdadmin login –l`	Not supported
Not supported	Not supported	`pdadmin logout`	Not supported
`ivadmin_objectspace_create()`	**PDProtObjectSpace.create ProtObjectSpace**	`pdadmin objectspace create` *objectspace_name*	**Object Space** → **Create Object Space** → fill in form → **Create**
`ivadmin_objectspace_delete()`	**PDProtObjectSpace.delete ProtObjectSpace**	`pdadmin objectspace delete` *objectspace_name*	**Object Space** → **Browse Object Space** → click object space name → **Delete**
`ivadmin_objectspace_list()`	**PDProtObjectSpace.list ProtObjectSpaces**	`pdadmin objectspace list`	**Object Space** → **Browse Object Space**
`ivadmin_pop_attach()`	**PDProtObject.attachPop** *PDProtObject object*.**attachPop**	`pdadmin pop attach` *object_name pop_name*	**POP** → **List POPss** → click POP name → **Attach** tab → **Attach** → type protected object path → **Attach**
`ivadmin_pop_attrdelkey()`	**PDPop.deleteAttribute** *PDPop object*.**deleteAttribute**	`pdadmin pop modify` *pop_name* `delete attribute` *attribute_name*	**POP** → **List POPs** → click POP name → **Extended Attributes** tab → select attributes → **Delete**
`ivadmin_pop_attrdelval()`	**PDPop.deleteAttributeValue** *PDPop object*.**deleteAttributeValue**	`pdadmin pop modify` *pop_name* `delete attribute` *attribute_name attribute_value*	Not supported
`ivadmin_pop_attrget()`	*PDPop object*.**getAttributeValues**	`pdadmin pop show` *pop_name* `attribute`	**POP** → **List POPs** → click POP name → **Extended Attributes** tab
`ivadmin_pop_attrlist()`	*PDPop object*.**getAttributeNames**	`pdadmin pop list` *pop_name* `attribute`	**POP** → **List POPs** → click POP name → **Extended Attributes** tab
`ivadmin_pop_attrput()`	**PDPop.setAttributeValuePDPop object.setAttributeValue**	`pdadmin pop modify` *pop_name* `set attribute` *attribute_name attribute_value*	**POP** → **List POPs** → click POP name → **Extended Attributes** tab → **Create** → fill in form → **Apply**
`ivadmin_pop_create()`	**PDPop.createPop**	`pdadmin pop create` *pop_name*	**POP** → **Create POP** → fill in form → **Create**
`ivadmin_pop_delete()`	**PDPop.deletePop**	`pdadmin pop delete` *pop_name*	**POP** → **List POPs** → select POP names → **Delete**

Table 29. Mapping of the administration C API to the Java methods, the pdadmin interface, and Web Portal Manager (continued)

C API	Java class and method	pdadmin command equivalent	Web Portal Manager equivalent
ivadmin_pop_detach()	**PDProtObject.detachPop** *PDProtObjectobject*.**attachPop**	pdadmin pop detach *object_name*	**POP** → **List POPs** → click POP name → **Attach** tab → select object → **Detach**
ivadmin_pop_find()	**PDProtObject.listProtObjectsByPop**	pdadmin pop find *pop_name*	**POP** → **List POPs** → click POP name → **Attach** tab
ivadmin_pop_get()	**PDPop** constructor	pdadmin pop show *pop_name*	**POP** → **List POPs** → click POP name
ivadmin_pop_getanyothernw2()	*PDPop object*.**getIPAuthInfo**	pdadmin pop show *pop_name*	**POP** → **List POPs** → click POP name
ivadmin_pop_getauditlevel()	*PDPop object*.**getAuditLevel**	pdadmin pop show *pop_name*	**POP** → **List POPs** → click POP name
ivadmin_pop_getdescription()	*PDPop object*.**getDescription**	pdadmin pop show *pop_name*	**POP** → **List POPs** → click POP name
ivadmin_pop_getid()	*PDPop object*.**getId**	pdadmin pop show *pop_name*	**POP** → **List POPs** → click POP name
ivadmin_pop_getipauth2()	*PDPop object*.**getIPAuthInfo**	pdadmin pop show *pop_name*	**POP** → **List POPs** → click POP name
ivadmin_pop_getqop()	*PDPop object*.**getQOP**	pdadmin pop show *pop_name*	**POP** → **List POPs** → click POP name
ivadmin_pop_gettod()	*PDPop object*.**getTodAccessInfo**	pdadmin pop show *pop_name*	**POP** → **List POPs** → click POP name
ivadmin_pop_getwarnmode()	*PDPop object*.**getWarningMode**	pdadmin pop show *pop_name*	**POP** → **List POPs** → click POP name
ivadmin_pop_list()	**PDPop.listPops**	pdadmin pop list	**POP** → **List POPs**
ivadmin_pop_list()	**PDPop.listPops**	pdadmin pop list *pop_name*	**POP** → **List POPs** → click POP name
ivadmin_pop_removeipauth2()	**PDPop.removeIPAuthInfo** *PDPop object*.**removeIPAuthInfo**	pdadmin pop modify *pop_name* set ipauth remove *network netmask*	**POP** → **List POPs** → click POP name → **IP Auth** tab → select IP authorization entries → **Delete**
ivadmin_pop_setanyothernw2()	**PDPop.setIPAuthInfo**	pdadmin pop modify *pop_name* set ipauth anyothernw *authentication_level*	**POP** → **List POPs** → click POP name → **IP Auth** tab → **Create** → select **Any Other Network** check box. and type the authentication level → **Create**
ivadmin_pop_setanyothernw_ forbidden2()	**PDPop.setIPAuthInfo**	pdadmin pop modify *pop_name* set ipauth anyothernw forbidden	**POP** → **List POPs** → click POP name → **IP Auth** tab → **Create** → select **Any Other Network** and **Forbidden** check boxes → **Create**
ivadmin_pop_setauditlevel()	**PDPop.setAuditLevelPDPop object .setAuditLevel**	pdadmin pop modify *pop_name* set audit-level {all | none | *audit_level_list*}	**POP** → **List POPs** → click POP name → select or clear appropriate check boxes → **Apply**
ivadmin_pop_setdescription()	**PDPop.setDescription** *PDPop object*.**setDescription**	pdadmin pop modify *pop_name* set description *description*	**POP** → **List POPs** → click POP name → modify description → **Apply**
ivadmin_pop_setipauth2()	**PDPop.setIPAuthInfo** *PDPop object*.**setIPAuthInfo**	pdadmin pop modify *pop_name* set ipauth add *network netmask authentication_level*	**POP** → **List POPs** → click POP name → **IP Auth** tab → **Create** → type the network, net mask, and authentication level → **Create**
ivadmin_pop_setipauth_forbidden2()	**PDPop.setIPAuthInfo** *PDPop object*.**setIPAuthInfo**	pdadmin pop modify *pop_name* set ipauth add *network netmask* forbidden	**POP** → **List POPs** → click POP name → **IP Auth** tab → **Create** → type network and net mask and select **Forbidden** check box → **Apply**
ivadmin_pop_setqop()	**PDPop.setQOP***PDPop object*.**setQOP**	pdadmin pop modify *pop_name* set qop {none | integrity | privacy}	**POP** → **List POPs** → click POP name → select appropriate quality of protection → **Apply**

Table 29. Mapping of the administration C API to the Java methods, the pdadmin interface, and Web Portal Manager (continued)

C API	Java class and method	pdadmin command equivalent	Web Portal Manager equivalent
`ivadmin_pop_settod()`	**PDPop.setTodAccessInfo** *PDPop object*.**setTodAccessInfo**	`pdadmin pop modify` *`pop_name`* `set tod-access {anyday | weekday |` *`day_list`*`}:{anytime |` *`time_spec-time_spec`*`} [:utc | local]`	**POP** → **List POPs** → click POP name → define time of day access → **Apply**
`ivadmin_pop_setwarnmode()`	**PDPop.setWarningMode** *PDPop object*.**setWarningMode**	`pdadmin pop modify` *`pop_name`* `set warning {yes | no]`	**POP** → **List POPs** → click POP name → select or clear **Warn Only On Policy Violation** check box → **Apply**
Not supported	Not supported	`pdadmin quit`	Not supported
`ivadmin_protobj_access()`	**PDProtObject.access**	`pdadmin object access` *`object_name permissions`*	Not supported.
`ivadmin_protobj_attachacl()`	**PDProtObject.attachAcl** *PDProtObject object*.**attachAcl**	`pdadmin acl attach` *`object_name acl_name`*	**ACL** → **List ACL** → click ACL name → **Attach** tab → **Attach** → type protected object path → **Attach**
`ivadmin_protobj_attachauthzrule()`	**PDProtObject.attachAuthzRul** *PDProtObject object*.**attachAuthzRule**	`pdadmin authzrule attach` *`object_name ruleid`*	**AuthzRule** → **List AuthzRule** → click authorization rule name → **Attach** tab → **Attach** → type protected object path → **Attach**
`ivadmin_protobj_attrdelkey()`	**PDProtObject.deleteAttribute** *PDProtObject object.deleteAttribute*	`pdadmin object modify` *`object_name`* `delete` *`attribute_name`*	**Object Space** → **Browse Object Space** → expand and click object name → **Extended Attributes** tab → select attribute → **Delete**
`ivadmin_protobj_attrdelval()`	**PDProtObject.deleteAttributeValue** *PDProtObject object.deleteAttributeValue*	`pdadmin object modify` *`object_name`* `delete` *`attribute_name attribute_value`*	Not supported
`ivadmin_protobj_attrget()`	*PDProtObject object*.**getAttributeValues**	`pdadmin object show` *`object_name`* `attribute` *`attribute_name`*	**Object Space** → **Browse Object Space** → expand and click object name → **Extended Attributes** tab
	PDProtObject.listProtObjects	`pdadmin object list`	**Object Space** → **Browse Object Space**
`ivadmin_protobj_attrlist()`	*PDProtObject object*.**getAttributeNames**	`pdadmin object list` *`object_name`* `attribute`	**Object Space** → **Browse Object Space** → expand and click object name → **Extended Attributes** tab
		`pdadmin object listandshow` *`object_name`*	Not supported
`ivadmin_protobj_attrput()`	**PDProtObject.setAttributeValue** *PDProtObject object.setAttributeValue*	`pdadmin object modify` *`object_name`* `set attribute` *`attribute_name attribute_value`*	**Object Space** → **Browse Object Space** → expand and click object name → **Extended Attributes** tab → **Create** → fill in form → **Apply**
`ivadmin_protobj_create()`	**PDProtObject.createProtObject**	`pdadmin object create` *`object_name description type`* `ispolicyattachable {yes|no}`	**Object Space** → **Create Object** → fill in form → **Create** The **type** field is not supported. You can select the **Can Policy be attached to this object** check box on the Protected Object Properties page.
`ivadmin_protobj_delete()`	**PDProtObject.deleteProtObject**	`pdadmin object delete` *`object_name`*	**Object Space** → **Browse Object Space** → expand and click object name → **Delete**
`ivadmin_protobj_detachacl()`	**PDProtObject.detachAcl** *PDProtObject object*.**detachAcl**	`pdadmin acl detach` *`object_name`*	**ACL** → **List ACL** → click ACL name → **Attach** tab → select protected object → **Detach**

Table 29. Mapping of the administration C API to the Java methods, the pdadmin interface, and Web Portal Manager (continued)

C API	Java class and method	pdadmin command equivalent	Web Portal Manager equivalent
`ivadmin_protobj_detachauthzrule()`	**PDProtObject.detachAuthzRule** *PDProtObject object*.**detachAuthzRule**	`pdadmin authzrule detach` *`object_name`*	**AuthzRule** → **List AuthzRule** → click authorization rule name → **Attach** tab → select object names → **Detach**
`ivadmin_protobj_exists()`	**PDProtObject.exists**	`pdadmin object exists` *`object_name`*	Not supported.
`ivadmin_protobj_get3()`	**PDProtObject** constructor **Note:** If the protected object name specified does not exist, default values are shown. To determine that a protected object exists, use the **`PDProtObject.exists`** command.	`pdadmin object show` *`object_name`*	**Object Space** → **Browse** **Object Space** → expand and click object name
`ivadmin_protobj_getaclid()`	*PDProtObject object*.**getAcl**	`pdadmin object show` *`object_name`*	**Object Space** → **Browse** **Object Space** → expand and click object name
`ivadmin_protobj_getauthzruleid()`	*PDProtObject object*.**getAuthzRule**	`pdadmin object show` *`object_name`*	**Object Space** → **Browse** **Object Space** → expand and click object name
`ivadmin_protobj_getdesc()`	*PDProtObject object*.**getDescription**	`pdadmin object show` *`object_name`*	**Object Space** → **Browse** **Object Space** → expand and click object name
`ivadmin_protobj_geteffaclid()`	*PDProtObject object*.**getEffectuveAclId**	`pdadmin object show` *`object_name`*	**Object Space** → **Browse** **Object Space** → expand and click object name
`ivadmin_protobj_geteffauthzruleid()`	*PDProtObject object* **.getEffectuveAuthzRuleId**	`pdadmin object show` *`object_name`*	**Object Space** → **Browse** **Object Space** → expand and click object name
`ivadmin_protobj_geteffpopid()`	*PDProtObject object*.**getEffectuvePopId**	`pdadmin object show` *`object_name`*	**Object Space** → **Browse** **Object Space** → expand and click object name
`ivadmin_protobj_getid()`	*PDProtObject object*.**getId**	`pdadmin object show` *`object_name`*	**Object Space** → **Browse** **Object Space** → expand and click object name
`ivadmin_protobj_getpolicyattachable()`	*PDProtObject object*.**isPolicyAttachable**	`pdadmin object show` *`object_name`*	**Object Space** → **Browse** **Object Space** → expand and click object name
`ivadmin_protobj_getpopid()`	*PDProtObject object*.**getPopId**	`pdadmin object show` *`object_name`*	**Object Space** → **Browse** **Object Space** → expand and click object name
	Not supported	`pdadmin object show` *`object_name`*	**Object Space** → **Browse** **Object Space** → expand and click object name
`ivadmin_protobj_list3()`	**PDProtObject.listProtObjects** **Note:** Before using this information and the product it supports, read the information in "Notices" on page 141.	`pdadmin object list` *`directory_name`*	**Object Space** → **Browse** **Object Space** → expand and click object name
`ivadmin_protobj_listbyacl()`	**PDProtObject.listProtObjectsByAcl**	`pdadmin acl find` *`acl_name`*	**ACL** → **List ACL** → click ACL name → **Attach** tab
`ivadmin_protobj_listbyauthzrule()`	**PDProtObject.listProtObjectsByAuthzRule**	`pdadmin authzrule find` *`ruleid`*	**AuthzRule** → **List AuthzRule** → click authorization rule name → **Attach** tab
`ivadmin_protobj_multiaccess()`	**PDProtObject.multiAccess**	`pdadmin object access` *`object_name permissions`*	Not supported.
`ivadmin_protobj_setdesc()`	**PDProtObject.setDescription** *PDProtObject object*.**setDescription**	`pdadmin object modify` *`object_name`* `set description` *`description`*	**Object Space** → **Browse** **Object Space** → expand and click object name → modify description → **Apply**
`ivadmin_protobj_setname()`	Not supported	`pdadmin object modify` *`object_name`* `name` *`name`* `conflict_resolution resolution_modifier`	Not supported

Table 29. Mapping of the administration C API to the Java methods, the pdadmin interface, and Web Portal Manager (continued)

C API	Java class and method	pdadmin command equivalent	Web Portal Manager equivalent
`ivadmin_protobj_ setpolicyattachable()`	**PDProtObject.setPolicyAttachable** *PDProtObject object*.**setPolicyAttachable**	`pdadmin object modify` *`object_name`* `isPolicyAttachable {yes|no}`	**Object Space** → **Browse Object Space** → expand and click object name → select or clear check box→ **Apply**
`ivadmin_protobj_settype()`	Not supported.	`pdadmin object modify` *`object_name`* `type` *`type`*	Not supported.
`ivadmin_response_getcode()`	*Not applicable.*	Not applicable.	Not applicable.
`ivadmin_response_getcount()`	*Not applicable.*	Not applicable.	Not applicable.
`ivadmin_response_getmessage()`	*Not applicable.*	Not applicable.	Not applicable.
`ivadmin_response_getmodifier()`	*Not applicable.*	Not applicable.	Not applicable.
`ivadmin_response_getok()`	*Not applicable.*	Not applicable.	Not applicable.
Not supported.	*Not supported.*	`pdadmin server list`	Not supported.
`ivadmin_server_gettasklist()`	**PDServer.getTaskList**	`pdadmin server listtasks` *`server_name`*	Not supported.
Not supported	*Not supported*	`pdadmin server show` *`server_name`*	Not supported.
`ivadmin_server_performtask()`	**PDServer.performTask**	`pdadmin server task` *`server_name server_task`*	Not supported. For more information about the WebSEAL server tasks and junction points, see the *IBM Security Access Manager for Web: WebSEAL Administration Guide*.
		`pdadmin server task` *`server_name`* `{help | stats | trace}`	Not supported.
`ivadmin_server_replicate()`	**PDServer.serverReplicate**	`pdadmin server replicate` *`server_name`*	Not supported.
`ivadmin_ssocred_create()`	**PDSSOCred.createSSOCred**	`pdadmin rsrccred create` *`resource_name`* `rsrcuser` *`resource_userid`* `rsrcpwd` *`resource_pwd`* `rsrctype {web | group} user` *`user_name`*	**User** → **Search Users** → **Search** → click user name → **GSO Credentials** tab → **Create** → fill in form → **Create**
`ivadmin_ssocred_create()`	**PDSSOCred.createSSOCred**	`pdadmin rsrccred create` *`resource_group_name`* `rsrcuser` *`resource_userid`* `rsrcpwd` *`resource_pwd`* `rsrctype {web | group} user` *`user_name`*	**User** → **Search Groups** → **Search** → click user name → **GSO Credentials** tab → **Create** → fill in form → **Create**
`ivadmin_ssocred_delete()`	**PDSSOCred.deleteSSOCred**	`pdadmin rsrccred delete` *`resource_name`* `rsrctype {web | group} user` *`user_name`*	**User** → **Search Users** → **Search** → click user name → **GSO Credentials** tab → select credentials → **Delete**
`ivadmin_ssocred_delete()`	**PDSSOCred.deleteSSOCred**	`pdadmin rsrccred delete` *`resource_group_name`* `rsrctype {web | group} user` *`user_name`*	**User** → **Search Groups** → **Search** → click user name → **GSO Credentials** tab → select credentials → **Delete**
`ivadmin_ssocred_get()`	**PDSSOCred** constructor	`pdadmin rsrccred show` *`resource_name`* `rsrctype {web | group} user` *`user_name`*	**User** → **Search Users** → **Search** → click user name → **GSO Credentials** tab
`ivadmin_ssocred_getid()`	*PDSSOCred object*.**getResourceName**	`pdadmin rsrccred show` *`resource_name`* `rsrctype {web | group} user` *`user_name`*	**User** → **Search Users** → **Search** → click user name → **GSO Credentials** tab
`ivadmin_ssocred_getssopassword()`	*PDSSOCred object*.**getResourcePassword**	Not applicable.	Not applicable.
`ivadmin_ssocred_getssouser()`	*PDSSOCred object*.**getResourceUser**	Not applicable.	Not applicable.

Table 29. Mapping of the administration C API to the Java methods, the pdadmin interface, and Web Portal Manager (continued)

C API	Java class and method	pdadmin command equivalent	Web Portal Manager equivalent
`ivadmin_ssocred_gettype()`	*PDSSOCred object*.**getResourceType**	`pdadmin rsrccred show` *resource_name* `rsrctype {web | group} user` *user_name*	**User** → **Search Users** → **Search** → click user name → **GSO Credentials** tab
`ivadmin_ssocred_getuser()`	*PDSSOCred object*.**getUser**	`pdadmin rsrccred show` *resource_name* `rsrctype {web | group} user` *user_name*	**User** → **Search Users** → **Search** → click user name → **GSO Credentials** tab
`ivadmin_ssocred_get()`	**PDSSOCred** constructor	`pdadmin rsrccred show` *resource_group_name* `rsrctype {web | group} user` *user_name*	**User** → **Search Groups** → **Search** → click user name → **GSO Credentials** tab
`ivadmin_ssocred_list()`	*PDSSOCred object*.**listAndShow SSOCreds** *PDSSOCred object*.**listSSOCreds**	`pdadmin rsrccred list user` *user_name*	**User** → **Search Users** → **Search** → click user name → **GSO Credentials** tab
`ivadmin_ssocred_set()`	**PDSSOCred.setSSOCred** *PDSSOCred object*.**setSSOCred**	`pdadmin rsrccred modify` *resource_name* `rsrctype {web | group} [-rsrcuser` *resource_userid*`] [-rsrcpwd` *resource_pwd*`] user` *user_name*	**User** → **Search Users** → **Search** → click user name → **GSO Credentials** tab → **Create** → modify form → **Create**
`ivadmin_ssocred_set()`	**PDSSOCred.setSSOCred** *PDSSOCred object*.**setSSOCred**	`pdadmin rsrccred modify` *resource_group_name* `rsrctype {web | group} [-rsrcuser` *resource_userid*`] [-rsrcpwd` *resource_pwd*`] user` *user_name*	**User** → **Search Groups** → **Search** → click user name → **GSO Credentials** tab → **Create** → modify form → **Create**
`ivadmin_ssogroup_addres()`	**PDSSOResourceGroup. addSSOResource** *PDSSOResourceGroup object* **addSSOResource**	`pdadmin rsrcgroup modify` *resource_group_name* `add rsrcname` *resource_name*	**GSO Resource** → **List GSO Groups** → select resource group → select members → **Add**
`ivadmin_ssogroup_create()`	**PDSSOResourceGroup. createSSOResourceGroup**	`pdadmin rsrcgroup create` *resource_group_name*	**GSO Resource** → **Create GSO Group** → fill in form → **Create**
`ivadmin_ssogroup_create()`	**PDSSOResourceGroup. createSSOResourceGroup**	`pdadmin rsrcgroup create` *resource_group_name* `-desc` *description*	**GSO Resource** → **Create GSO Group** → fill in form and modify the description → **Create**
`ivadmin_ssogroup_delete()`	**PDSSOResourceGroup. deleteSSOResourceGroup**	`pdadmin rsrcgroup delete` *resource_group_name*	**GSO Resource** → **List GSO Groups** → select resource groups → **Delete**
`ivadmin_ssogroup_get()`	**PDSSOResourceGroup** constructor	`pdadmin rsrcgroup show` *resource_group_name*	**GSO Resource** → **List GSO Groups** → select resource group
`ivadmin_ssogroup_getdescription()`	*PDSSOResourceGroup object*.**getDescription**	`pdadmin rsrcgroup show` *resource_group_name*	**GSO Resource** → **List GSO Groups** → select resource group
`ivadmin_ssogroup_getid()`	*PDSSOResourceGroup object*.**getId**	`pdadmin rsrcgroup show` *resource_group_name*	**GSO Resource** → **List GSO Groups** → select resource group
`ivadmin_ssogroup_getresources()`	*PDSSOResourceGroup object*.**getSSOResources**	`pdadmin rsrcgroup show` *resource_group_name*	**GSO Resource** → **List GSO Groups** → select resource group
`ivadmin_ssogroup_list()`	**PDSSOResourceGroup. listSSOResourceGroups**	`pdadmin rsrcgroup list`	**GSO Resource** → **List GSO Groups**
`ivadmin_ssogroup_removeres()`	**PDSSOResourceGroup. removeSSOResource** *PDSSOResourceGroup object* **.removeSSOResource**.	`pdadmin rsrcgroup modify` *resource_group_name* remove rsrcname *resource_name*	**GSO Resource** → **List GSO Groups** → select resource group → select members → **Remove**
`ivadmin_ssoweb_create()`	**PDSSOResource.createSSOResource**	`pdadmin rsrc create` *resource_name*	**GSO Resource** → **Create GSO** → fill in form → **Create**
`ivadmin_ssoweb_create()`	**PDSSOResource.createSSOResource**	`pdadmin rsrc create` *resource_name* `-desc` *description*	**GSO Resource** → **Create GSO** → fill in form and modify the description → **Create**

Table 29. Mapping of the administration C API to the Java methods, the pdadmin interface, and Web Portal Manager (continued)

C API	Java class and method	pdadmin command equivalent	Web Portal Manager equivalent
`ivadmin_ssoweb_delete()`	**PDSSOResource.deleteSSOResource**	`pdadmin rsrc delete` *resource_name*	**GSO Resource** → **List GSO** → select resources → **Delete**
`ivadmin_ssoweb_get()`	**PDSSOResource** constructor	`pdadmin rsrc show` *resource_name*	**GSO Resource** → **List GSO** → click resource
`ivadmin_ssoweb_getdescription()`	*PDSSOResource object*.**getDescription**	`pdadmin rsrc show` *resource_name*	**GSO Resource** → **List GSO** → click resource
`ivadmin_ssoweb_getid()`	*PDSSOResource object*.**getId**	`pdadmin rsrc show` *resource_name*	**GSO Resource** → **List GSO** → click resource
`ivadmin_ssoweb_list()`	**PDSSOResource.listSSOResources**	`pdadmin rsrc list`	**GSO Resource** → **List GSO**
`ivadmin_user_create3()`	**PDUser.createUser**	`pdadmin user create [-gsouser] [-no-password-policy]` *user_name dn cn sn password* [*group1* [*group2* ...]]	**User** → **Create User** → fill in form → **Create**
`ivadmin_user_delete2()`	**PDUser.deleteUser**	`pdadmin user delete [-registry]` *user_name*	**User** → **Search Users** → type pattern and maximum results → **Search** → select user names → **Delete**
`ivadmin_user_get()`	**PDUser** constructor	`pdadmin user show` *user_name*	**User** → **Search Users** → type pattern and maximum results → **Search** → click user name
`ivadmin_user_getaccexpdate()`	*PDPolicy object*.**getAcctExpDate**	`pdadmin user get account-expiry-date -user` *user_name*	**User** → **Search Users** → type pattern and maximum results → **Search** → click user name → **Policy** tab
`ivadmin_user_getaccountvalid()`	*PDUser object*.**isAccountValid**	`pdadmin user show` *user_name*	**User** → **Search Users** → type pattern and maximum results → **Search** → click user name
`ivadmin_user_getbydn()`	**PDUser** constructor	`pdadmin user show-dn` *dn*	Not supported
`ivadmin_user_getcn()`	*PDUser object*.**getFirstName**	`pdadmin user show` *user_name*	**User** → **Search Users** → type pattern and maximum results → **Search** → click user name
`ivadmin_user_getdescription()`	*PDUser object*.**getDescription**	`pdadmin user show` *user_name*	**User** → **Search Users** → type pattern and maximum results → **Search** → click user name
ivadmin_user_getdisabletimeint()	*PDPolicy object*.**getAcctDisableTimeInterval**	`pdadmin policy get disable-time-interval -user` *user_name*	**User** → **Search Users** → type pattern and maximum results → **Search** → click user name → **Policy** tab
ivadmin_user_getdn()	***PDUser object***.getRgyName	`pdadmin user show` *user_name*	**User** → **Search Users** → type pattern and maximum results → **Search** → click user name
ivadmin_user_getid()	*PDUser object*.**getId**	`pdadmin user show` *user_name*	**User** → **Search Users** → type pattern and maximum results → **Search** → click user name
ivadmin_user_getmaxconcurwebsess()	*PDPolicy object* **.getMaxconcurrentWebSessions**	`pdadmin policy get max-concurrent-web-sessions -user` *user_name*	**User** → **Show Global User Policy** → view `Max Concurrent Web Sessions`
ivadmin_user_getmaxlgnfails()	*PDPolicy object*.**getMaxFailedLogins**	`pdadmin policy get max-login-failures -user` *user_name*	**User** → **Search Users** → type pattern and maximum results → **Search** → click user name → **Policy** tab
ivadmin_user_getmaxpwdage()	*PDPolicy object*.**getMaxPwdAge**	`pdadmin policy get max-password-age -user` *user_name*	**User** → **Search Users** → type pattern and maximum results → **Search** → click user name → **Policy** tab
ivadmin_user_getmaxpwdrepchars()	*PDPolicy object*.**getMaxPwdRepChars**	`pdadmin policy get max-password-repeated-chars -user` *user_name*	**User** → **Search Users** → type pattern and maximum results → **Search** → click user name → **Policy** tab

Table 29. Mapping of the administration C API to the Java methods, the pdadmin interface, and Web Portal Manager (continued)

C API	Java class and method	pdadmin command equivalent	Web Portal Manager equivalent
ivadmin_user_getmemberships()	*PDUser object*.**getGroups**	`pdadmin user show-groups` *user_name*	**User** → **Search Users** → type pattern and maximum results → **Search** → click user name → **Groups** tab
ivadmin_user_getminpwdalphas()	*PDPolicy object*.**getMinPwdAlphas**	`pdadmin policy get min-password-alphas -user` *user_name*	**User** → **Search Users** → type pattern and maximum results → **Search** → click user name → **Policy** tab
ivadmin_user_getminpwdlen()	*PDPolicy object*.**getMinPwdLen**	`pdadmin policy get min-password-length -user` *user_name*	**User** → **Search Users** → type pattern and maximum results → **Search** → click user name → **Policy** tab
ivadmin_user_getminpwdnonalphas()	*PDPolicy object*.**getMinPwdNonAlphas**	`pdadmin policy get min-password-non-alphas -user` *user_name*	**User** → **Search Users** → type pattern and maximum results → **Search** → click user name → **Policy** tab
`ivadmin_user_getpasswordvalid()`	*PDUser object*.**isPasswordValid**	`pdadmin user show` *user_name*	**User** → **Search Users** → type pattern and maximum results → **Search** → click user name
`ivadmin_user_getpwdspaces()`	*PDPolicy object*.**pwdSpacesAllowed**	`pdadmin policy get password-spaces -user` *user_name*	**User** → **Search Users** → type pattern and maximum results → **Search** → click user name → **Policy** tab
`ivadmin_user_getsn()`	*PDUser object*.**getLastName**	`pdadmin user show` *user_name*	**User** → **Search Users** → type pattern and maximum results → **Search** → click user name
Not applicable	*PDUser object*.**isPDUser**	`pdadmin user show` *user_name*	**User** → **Search Users** → type pattern and maximum results → **Search** → click user name
`ivadmin_user_getssouser()`	*PDUser object*.**isSSOUser**t_memory_error	`pdadmin user show` *user_name*	**User** → **Search Users** → type pattern and maximum results → **Search** → click user name
`ivadmin_user_gettodaccess()`	*PDPolicy object*.**getAccessibleDays** *PDPolicy object*.**getAccessStartTime** *PDPolicy object*.**getAccessEndTime**	`pdadmin policy get tod-access -user` *user_name*	**User** → **Search Users** → type pattern and maximum results → **Search** → click user name → **Policy** tab
`ivadmin_user_import2()`	**PDUser.importUser**	`pdadmin user import [-gsouser]` *user_name dn* `[`*group_name*`]`	**User** → **Import User** → fill in form → **Create**
`ivadmin_user_list()`	**PDUser.listUsers**	`pdadmin user list` *pattern max_return*	**User** → **Search Users** → type pattern and maximum results → **Search**
`ivadmin_user_listbydn()`	**PDUser.listUsers**	`pdadmin user list-dn` *pattern max_return*	Not supported.
`ivadmin_user_setaccexpdate()`	**PDPolicy.setAcctExpDate** *PDPolicy object*.**setAcctExpDate**	`pdadmin policy set account-expiry-date {unlimited |` *absolute_time* `| unset} -user` *user_name*	**User** → **Search Users** → type pattern and maximum results → **Search** → click user name → **Policy** tab → modify value → **Apply**
`ivadmin_user_setaccountvalid()`	**PDUser.setAccountValid** *PDUser object*.**setAccountValid**	`pdadmin user modify` *user_name* `account-valid {yes|no}`	**User** → **Search Users** → type pattern and maximum results → **Search** → click user name → select or clear check box → **Apply**
`ivadmin_user_setdescription()`	**PDUser.setDescription** *PDUser object*.**setDescription**	`pdadmin user modify` *user_name* `description` *description*	**User** → **Search Users** → type pattern and maximum results → **Search** → click user name → modify description → **Apply**
`ivadmin_user_setdisabletimeint()`	**PDPolicy.setAcctDisableTime** *PDPolicy object*.**setAcctDisableTime**	`pdadmin policy set disable-time-interval {`*number* `| unset | disable} -user` *user_name*	**User** → **Search Users** → type pattern and maximum results → **Search** → click user name → **Policy** tab → modify value → **Apply**

Table 29. Mapping of the administration C API to the Java methods, the pdadmin interface, and Web Portal Manager (continued)

C API	Java class and method	pdadmin command equivalent	Web Portal Manager equivalent
`ivadmin_user_setmaxconcurwebsess()`	**PDPolicy.MaxConcurrentWebSessions Displaced** **PDPolicy.MaxConcurrentWebSessions Unlimited** **PDPolicy.MaxConcurrentWebSessions Enforced** **PDPolicy.setMaxconcurrent WebSessions** *PDPolicy object*.**setMaxconcurrent WebSessions**	`pdadmin policy set max-concurrent-web-sessions {`*number* | `displace | unlimited | unset} -user` *user_name*	**User** → **Search Users** → type pattern and maximum results → **Search** → click user name → **Policy** tab → modify value → **Apply**
`ivadmin_user_setmaxlgnfails()`	**PDPolicy.setMaxFailedLogins** *PDPolicy object*.**setMaxFailedLogins**	`pdadmin policy set max-login-failures {`*number* | `unset} -user` *user_name*	**User** → **Search Users** → type pattern and maximum results → **Search** → click user name → **Policy** tab → modify value → **Apply**
`ivadmin_user_setmaxpwdage()`	**PDPolicy.setMaxPwdAge** *PDPolicy object*.**setMaxPwdAge**	`pdadmin policy set max-password-age {unset |` *relative_time*`} -user` *user_name*	**User** → **Search Users** → type pattern and maximum results → **Search** → click user name → **Policy** tab → modify value → **Apply**
`ivadmin_user_setmaxpwdrepchars()`	**PDPolicy.setMaxPwdRepChars** *PDPolicy object*.**setMaxPwdRepChars**	`pdadmin policy set max-password-repeated-chars {`*number* | `unset} -user` *user_name*	**User** → **Search Users** → type pattern and maximum results → **Search** → click user name → **Policy** tab → modify value → **Apply**
`ivadmin_user_setminpwdalphas()`	**PDPolicy.setMinPwdAlphas** ***PDPolicy object***.setMinPwdAlphas	`pdadmin policy set min-password-alphas {`*number* | `unset} -user` *user_name*	**User** → **Search Users** → type pattern and maximum results → **Search** → click user name → **Policy** tab → modify value → **Apply**
`ivadmin_user_setminpwdlen()`	**PDPolicy.setMinPwdLen** *PDPolicy object*.**setMinPwdLen**	`pdadmin policy set min-password-length {`*number* | `unset} -user` *user_name*	**User** → **Search Users** → type pattern and maximum results → **Search** → click user name → **Policy** tab → modify value → **Apply**
`ivadmin_user_setminpwdnonalphas()`	**PDPolicy.setMinPwdNonAlphas** *PDPolicy object*.**setMinPwdNonAlphas**	`pdadmin policy set min-password-non-alphas {`*number* | `unset} -user` *user_name*	**User** → **Search Users** → type pattern and maximum results → **Search** → click user name → **Policy** tab → modify value → **Apply** tab
`ivadmin_user_setpassword()`	**PDUser.setPassword** *PDUser object*.**setPassword**	`pdadmin user modify` *user_name* `password` *password*	**User** → **Search Users** → type pattern and maximum results → **Search** → click user name → modify password → **Apply**
`ivadmin_user_setpasswordvalid()`	**PDUser.setPasswordValid** *PDUser object*.**setPasswordValid**	`pdadmin user modify` *user_name* `password-valid {yes|no}`	**User** → **Search Users** → type pattern and maximum results → **Search** → click user name → select or clear check box → **Apply**
`ivadmin_user_setpwdspaces()`	**PDPolicy.setPwdSpacesAllowed** ***PDPolicy object***.setPwdSpacesAllowed	`pdadmin policy set password-spaces {yes | no | unset} -user` *user_name*	**User** → **Search Users** → type pattern and maximum results → **Search** → click user name → **Policy** tab → modify value → **Apply**
`ivadmin_user_setssouser()`	**PDUser.setSSOUser** *PDUser object*.**setSSOUser**	`pdadmin user modify` *user_name* `gsouser {yes|no}`	**User** → **Search Users** → type pattern and maximum results → **Search** → click user name → select or clear check box → **Apply**
`ivadmin_user_settodaccess()`	**PDPolicy.setTodAccess** *PDPolicy object*.**setTodAccess**	`pdadmin policy set tod-access` *tod_value* `-user` *user_name*	**User** → **Search Users** → type pattern and maximum results → **Search** → click user name → **Policy** tab → modify value → **Apply**

Appendix D. Registry Direct Java API

The Registry Direct API directly accesses the underlying Security Access Manager registry rather than through Authorization servers or Policy Server.

This API also provides access to most of the underlying registry user attributes and the attributes available through the traditional Security Access Manager Java API.

The advantages of this API are as follows:

- Removes the dependency on the Policy Server, a single point of failure.
- Provides more attribute access for developers.
- Improves performance and scalability.

This API provides attribute read-only Global Sign On (Single Sign On resource credential) support. It does not create, enable, disable, or delete the users that are enabled for Global Sign On.

Design

The Registry Direct API is a set of Java interfaces that provide the required administration and authentication methods. This design supports only IBM Tivoli® Directory Manager registry types.

The factory class generates an instance that implements the primary interface for specific registry.

Use the provided utility class to configure the API configuration in the stand-alone mode. The Security Access Manager Java API configuration class is for both the Security Access Manager Java API and Registry Direct Java API. If authorization is enabled in the Registry Direct Java API, the Registry Direct API uses the existing Java API to authorize access to API methods.

Security Access Manager Java API

This section illustrates how the API works.

The Security Access Manager Java API uses the Policy Server for user, group, and policy administration. The API uses the Authorization Server for authentication.

The following diagram illustrates the Security Access Manager Java API.

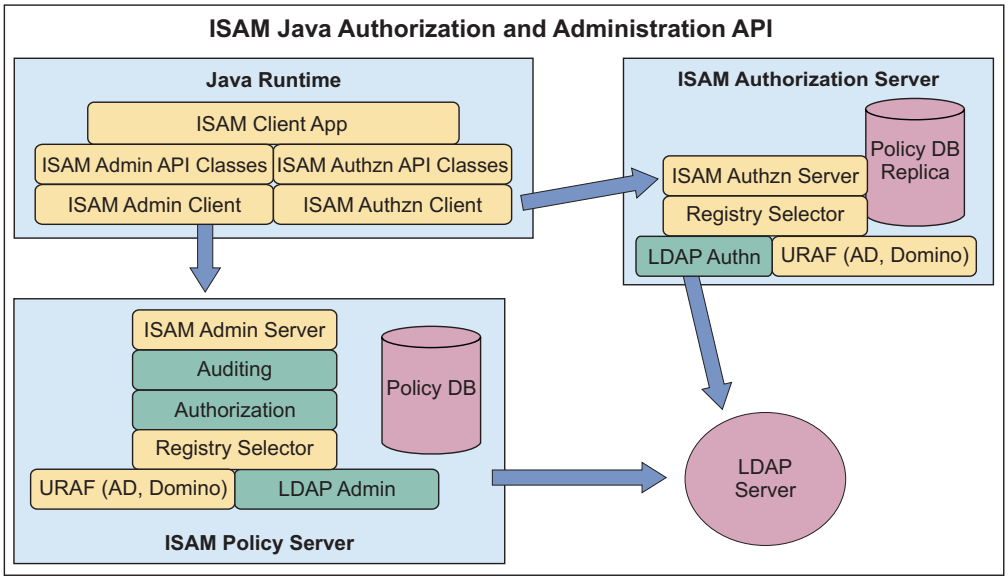

Figure 5. Security Access Manager Java API

Registry Direct Java API

If authorization is enabled, the Security Access Manager Java API requires the Authorization Server. To reduce this dependency, run the Security Access Manager Java API in local mode.

In local mode, the Registry Direct API authorizes decisions internally instead of using the Authorization server. The Authorization Server generates a credential for the administrator user who was authorized when this API starts.

The following diagram illustrates the Registry Direct Java API.

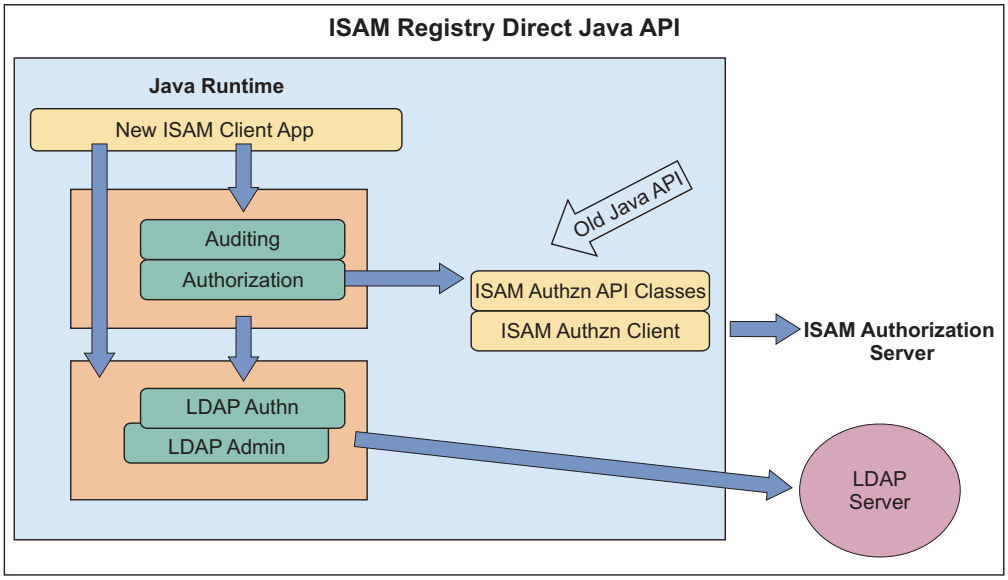

Figure 6. Registry Direct Java API

Published API

The API published to users consists of Java Interfaces and Factory classes to generate instances of the primary interface.

See detailed information about each class in the Javadoc included with the product. The following sections describe the Java interfaces and methods of the API.

com.tivoli.pd.rgy.RgyRegistry

The primary interface `RgyRegistry` consists of the following methods:

- getUser(), getNativeUser(), getGroup(), and getNativeGroup()
- createUser() and createGroup()
- newRgyAttributes()
- deleteUser() and deleteGroup()
- listUsers(), listNativeUsers(), listGroups, and listNativeGroups()
- setRgyThreadLocale() and getRgyThreadLocale()
- toRegistryDate() and fromRegistryDate()
- condenseResourceCredential() and expandResourceCredential()
- close()

getUser(), getNativeUser(), getGroup(), and getNativeGroup()

Fetch Security Access Manager and native user or group attributes. They create corresponding `RgyUser` or `RgyGroup` instances to hold these attributes and allow operations on the entity.

For `getNativeUser()` or `getNativeGroup()`, the native user or group uses the native ID (a DN for LDAP native registries), and the attributes of the user or group are fetched.

If the native user or group is also a Security Access Manager entity, the corresponding Security Access Manager attributes are also fetched.

When you import the native entity, the API uses the Security Access Manager domain that is passed through the `getNativeUser()` or `getNativeGroup()`

An `RgyUser` or `RgyGroup` instance does not need to be a Security Access Manager entity and might contain only the native entity attributes. You can examine the virtual attribute `isSecEntity` to determine whether the entity exists. If the value of `isSecEntity` is `true`, the Security Access Manager entity exists.

Prototype:

```
RgyUser getUser(String domain, String userId) throws RgyException;
RgyUser getNativeUser(String domain, String userNativeId) throws
RgyException;
RgyGroup getGroup(String domain, String groupId) throws RgyException;
RgyGroup getNativeGroup(String domain, String groupNativeId) throws
RgyException;
```

createUser() and createGroup()

Create a Security Access Manager user or group along with the underlying native entity. If the underlying native entity exists, use `getNativeUser()` or `getNativeGroup()` with the `RgyUser` or `RgyGroup` `import()` method.

Supply the required attributes like `cn` and `sn`.

Optionally, specify the following attributes during method creation:

- Native entity attributes such as `description`.
- Security Access Manager user attributes like `secAcctValid`.
- Security Access Manager user policy attributes such as `passwordMinLength`.

Prototype:

```
RgyUser createUser(String domain, String userId,
String userNativeId, char[] password,
boolean bypassPasswordPolicy,
RgyAttributes rgyAttributes,
Set <string> groupIds)throws RgyException;
RgyGroup createGroup(String domain, String groupId,
String groupNativeId,
RgyAttributes rgyAttributes)throws RgyException;
```

newRgyAttributes()

Creates an empty instance of `RgyAttributes`. This method populates and supplies the empty instance to `createUser()`, `createGroup()`, `RgyUser.import()`, and `RgyGroup.import()` methods.

Prototype:

```
RgyAttributes newRgyAttributes();
```

deleteUser() and deleteGroup()

Deletes Security Access Manager user and group, and optionally deletes the native entity.

This API does not update the GSO data of the user. You cannot delete GSO enabled users. You can use the virtual attribute `isGSOUser` to confirm if an `RgyUser` is GSO enabled.

The `WarningNativeEntityInUseRgyException` error is generated if other applications create entries under the native entry. Despite this error, the API deletes this Security Access Managerentity. You can ignore the error if the method permits.

Prototype:

```
void deleteUser(String domain, String userId, boolean
deleteNativeUser) throws RgyException;

void deleteGroup(String domain, String groupId, boolean
deleteNativeGroup) throws RgyException;
```

listUsers(), listNativeUsers(), listGroups, and listNativeGroups()

`listUsers()` lists Security Access Manager users

`listNativeUsers()` lists the native users

`listgroups()` lists the Security Access Manager groups

`listNativegroup()` lists the native groups.

The native lists return native IDs. For LDAP, the native id is a DN. Each native entity, the ID refers contains an attribute that matches the supplied attribute pattern. The non-native list methods returns Security Access

Manager IDs that match the supplied pattern. All methods return an RgyIterator instance to iterate the result set.

The PageSize parameter is optional and it can be ignored by the API.

Prototype:

```
RgyIterator listUsers(String domain, String userIdPattern,
int maxResults,int pageSize)throws RgyException;
RgyIterator listNativeUsers(String
searchAttributeName, String searchAttributePattern,   int maxResults,
int pageSize) throws RgyException;
RgyIterator listGroups(String domain, String groupIdPattern,
int maxResults,int pageSize)throws RgyException;
RgyIterator listNativeGroups(String searchAttributeName, String
searchAttributePattern,int maxResults, int pageSize)
throws RgyException;
```

setRgyThreadLocale() and getRgyThreadLocale()

Set and fetch the locale that is used when generating messages for RgyExceptions. You can set locale independently for each thread. If the set method is not invoked for a thread, the default locale for that thread is Java Runtime default locale. The log messages use the Java Runtime default locale and are not affected by setRgyThreadLocale() method.

Prototype:

```
void setRgyThreadLocale(Locale locale);
Locale getRgyThreadLocale();
```

toRegistryDate() and fromRegistryDate()

toRegistryDate() converts a Java date instance into a string format. You can supply this string format to the API for Security Access Manager attributes that require an absolute date, such as secAcctExpires.

fromRegistryDate() to interpret the date that is provided in Security Access Manager attributes such as secPwdLastChanged and secPwdLastUsed.

Prototype:

```
String toRegistryDate(Date javaDate) throws RgyException;
Date fromRegistryDate(String registryDate) throws RgyException;
```

condenseResourceCredential() and expandResourceCredential()

The expandResourceCredential method extracts the four component values from condensed form into an array of strings. The components are indexed as follows:

```
public final int RESOURCE_CRED_NAME_INDEX = 0;
public final int RESOURCE_CRED_TYPE_INDEX = 1;
public final int RESOURCE_CRED_USER_INDEX = 2;
public final int RESOURCE_CRED_PASSWORD_INDEX = 3;
public final int RESOURCE_CRED_LENGTH = 4;
```

CondenseResourceCredential() reverses the processes.

Each value of the attribute resourceCredentials is returned from the registry for users in condensed form.

The component value that is indexed by RESOURCE_CRED_TYPE_INDEX is one of the following:

```
public static final String RESOURCE_TYPE_WEB_VALUE = "Web Resource";
public static final String RESOURCE_TYPE_GROUP_VALUE = "Resource Group";
```

close()

Releases any used resources, such as open connections to LDAP. The RgyRegistry methods and any instances of RgyUser, RgyGroup, or RgyIterator generated by the RgyRegistry instance must not be used after the instance is closed.

Prototype:

```
void close();
```

com.tivoli.pd.rgy.RgyEntity

RgyEntity provides methods common to RgyUser and RgyGroup interfaces.

This section explains the following methods:

- getId() and getNativeId()
- attributeNameIterator(), getOneAttributeValue(), and getAttributeValues()
- attributeDelete(), attributeReplace(), and attributeAdd()
- getDomain() and getRgyRegistry()

getId() and getNativeId()

Provide the Security Access Manager ID for the user or group with getId() and the native registry ID with getNativeId(). This product supports only LDAP registries. The registry native ID is an LDAP Distinguished Name. The value of Security Access Manager ID is null if the LDAP account does not have any associated Security Access Manager identity.

Example :

- getId() returns a string similar to sec_master.
- getNativeId() returns a string similar to cn=SecurityMaster,secAuthority=Default.

Prototype:

```
String getId();
String getNativeId();
```

attributeNameIterator(), getOneAttributeValue(), and getAttributeValues()

You can obtain available attribute names using the attributeNameIterator() method.

getOneAttributeValue() is a convenience method you can use when you are sure that the attribute has only one value.

The getAttributeValues() Security Access Manager returns the attribute values as an array of objects.

All the value objects in the array are in the same class as a string or byte array. The cached values in the RgyEntity for the attributes are never read again from the registry. You must fetch a new instance of the entity using the RgyRegistry to ensure that the attribute values are the latest.

Prototype:

```
Iterator <String> attributeNameIterator();
Object getOneAttributeValue(String name);
Object[] getAttributeValues(String name);
```

attributeDelete(), attributeReplace(), and attributeAdd()

Modify the attributes of entities with these methods. Attribute values are either a string or byte. If an attribute has multiple values, you can supply

all the values as objects. All objects in the array can represent the same class. The API updates the values of both `RgyEntity` and the attributes from the cache to the registry.

Prototype: void attributeDelete(String name) throws RgyException; void attributeDelete(String name, Object value) throws RgyException; void attributeDelete(String name, Object[] values) throws RgyException; void attributeReplace(String name, Object value) throws RgyException; void attributeReplace(String name, Object[] values) throws RgyException; void attributeAdd(String name, Object value) throws RgyException; void attributeAdd(String name, Object[] values) throws RgyException;

Note: The Delete, Replace, or Add method of the `RgyEntity` updates the cached values only for the instance on which the method is invoked.

getDomain() and getRgyRegistry()

`getDomain()` returns a specific domain when `RgyRegistry` fetches the entity.

`getRgyRegistry()` returns the `RgyRegistry` instance that instantiated the `RgyEntity` instance.

com.tivoli.pd.rgy.RgyUser

`RgyUser` extends `RgyEntity` to provide user-specific methods.

The following section explains the methods in the `RgyUser` interface:

- authenticate()
- changePassword()
- setPassword()
- listGroups() and listNativeGroups()
- importNativeUser()

authenticate()

Fetch the policy and account state attribute values from `RgyUser` to ensure that the values used during authentication are the latest cached values in the `RgyUser` instance. This method does not generate a Security Access Manager credential.

If the authentication is successful, no error is generated. Otherwise, an error which indicates the reason for the failure is generated. Failure might be caused by a wrong password or any other factors, such as an unavailable account.

The Security Access Manager password validation policy is based on policies and account states. The Registry Direct Java API password validation process is compatible with the Security Access Manager password validation process.

Prototype:

```
void authenticate(char[] password) throws RgyException;
```

Note: Authentication takes the time of day access restriction into account. When setting a password policy, the user might provide a list of days, start time, and end time. The start time and end time apply to each day on the list. If the specified start time is greater than the specified end time, then the access is allowed until the specified end time of the next day.

changePassword()

Authenticates the current password and, if successful, sets the password to the new value.

If the authentication of the current password succeeds, the API sets the new password value. If the configuration property `ldap.enhanced-pwd-policy` is enabled, the password is updated by using the users credential. This method supports the native LDAP policy, which requires users to change the password after the administrator resets the password. Use the `setPassword()` method to reset the administrative password.

Prototype:

```
void changePassword(char[] currentPassword, char[] newPassword)
throws RgyException;
```

setPassword()

Sets the account password to the new value.

This method updates the user password by using the administrative account credentials of the Registry Direct API. If a specific native LDAP policy is enabled for the account, this method resets the native registry account state.

Note: Use this method when the administrator resets the user password, or the user password is reset by using the user-self-care password recovery process.

Prototype:

```
void setPassword(char[] newPassword) throws RgyException;
```

listGroups() and listNativeGroups()

`listGroups()` lists the groups to which the user belongs.

`listNativeGroups()` method returns a list of the native IDs of the groups. The list might include groups that are not Security Access Manager enabled. The group list is not cached in the `RgyUser` instance, and each invocation of the methods searches the registry to determine the membership.

Prototype:

```
Set <string> listGroups() throws RgyException;
Set <string> listNativeGroups() throws RgyException;
```

importNativeUser()

Converts the LDAP native user account into a Security Access Manager entity.

Prototype:

```
void importNativeUser(String userId, RgyAttributes rgyAttributes,
String groupId) throws RgyException;
```

com.tivoli.pd.rgy.RgyGroup

`RgyGroup` extends `RgyEntity` to provide group-specific methods.

This section describes the following methods in the `RgyGroup` interface:

- listMemberIds() and listMemberNativeIds()
- addMembers() and removeMembers()
- importNativeGroup()

listMemberIds() and listMemberNativeIds()

Returns a list of members who belong to the group.

When you call these methods, they fetch the member list directly from the registry such as LDAP. The member list is not cached in the `RgyGroup` instance.

`ListMemberNativeIds()` returns a list of native IDs (DNs for LDAP). The returned membership list can include native IDs of users who are not Security Access Manager enabled.

Prototype:

```
Set <string> listMemberIds() throws RgyException;
Set <string> listMemberNativeIds() throws RgyException;
```

addMembers() and removeMembers()

Adds and removes Security Access Manager users from the group membership list.

This method does not provide any option to manage the membership of dynamic or nested groups. These methods fail if the membership list is determined by dynamic methods or from nested group membership.

Prototype:

```
void addMembers(List <String> memberIds) throws RgyException;
void removeMembers(List <String> memberIds) throws RgyException;
```

importNativeGroup()

Converts the LDAP native group into a Security Access Manager entity.

Prototype:

```
void importNativeGroup(String groupId, RgyAttributes rgyAttributes)
throws RgyException;
```

com.tivoli.pd.rgy.RgyIterator

This interface provides an iterator for lists of user and group IDs.

Depending on the `RgyRegistry` method that returns the `RgyIterator` instance, the group IDs are either Security Access Manager IDs or native IDs.

This section explains the following methods in the `RgyIterator` interface:

- hasNext()
- next()
- close()

hasNext()

Returns the Boolean value `true` if another ID is available.

This method generates **SizeLimitExceededRgyException** if more results are available than those specified in `maxResults` when the `RgyIterator` was constructed, or the Registry Server is configured to allow. This exception occurs on the call after returning the last available ID.

Prototype:

```
boolean hasNext() throws RgyException;
```

next()

Returns the next available ID.

Prototype:

```
String next() throws RgyException;
```

close()

Stops the iteration.

If `RgyIterator` does not throw an exception or `hasNext()` does not return `false` and the caller has finished using the `RgyIterator` instance, call `close()` immediately to release any used resource. Each open `RgyIterator` instance opens a connection to the native registry.

The Registry Direct Java API limits the number of open `RgyIterator` to restrict the number of simultaneous connections. When the maximum limit is reached, instantiating new `RgyIterator` is not possible until at least one of the existing connections is closed.

Prototype:

```
void close();
```

com.tivoli.pd.rgy.ldap.RgyAttributes

`RgyAttributes` creates a collection of attributes for creating or importing a user or group.

This section explains the following methods available in the `RgyAttributes` interface:

- putAttribute()
- addAttribute()
- removeAttribute()
- putAttributesInto()
- getOneAttributeValue() and getAttributeValues()
- nameIterator()

putAttribute()

Replaces any existing attribute value with the specified values. If the specified attribute does not exist in the `RgyAttribute` instance, it is created by using the values that are passed through this method.

addAttribute()

Adds one or more values to the existing values for the specified attribute. If the specified attribute does not exist, it is created by using the values that are passed through this method.

removeAttribute()

Removes the specified values from the attribute that is passed through the `RgyAttributes` instance.

If the specified attribute is not present in the `RgyAttributes` instance, it is ignored.

If the attribute does not have any associated values, the attribute is removed after the attributes values are removed.

If only the attribute name is specified, this method removes the attribute completely.

putAttributesInto()

All attributes available in the`RgyAttribute` are added to the specified `RgyAttribute` instance. This method replaces the attributes that contain the same name.

getOneAttributeValue() and getAttributeValues()

Fetches the values of the named attribute. If the attribute named by `getOneAttributeValue()` contains more than one value, any one attribute value is returned. There is no specific pattern that is based on which the method selects the value.

nameIterator()

Returns an `Iterator<String>` that provides the names of all the attributes currently stored in the `RgyAttributes` instance.

com.tivoli.pd.rgy.ldap.LdapRgyRegistryFactory

This factory creates instances of the API interfaces that manipulate Security Access Manager entities in LDAP registries.

When configured correctly, the factory authorizes and audits the API methods. This section explains the following methods available in the `LdapRgyRegistryFactory` interface:

- getRgyRegistryInstance()
- getLdapRgyRegistryInstance()

getRgyRegistryInstance()

The primary method obtains `RgyRegistry` instance for LDAP registries.

Consider the other methods only if it requires authorization and the caller provides `PDAuthorizationContext` instance this API uses for authorization checks. This instance is required if `PDAuthorizationContext` is shared for other purposes because only one `PDAuthorizationContext` must be instantiated per configuration file that is created by the `com.tivoli.pd.jcfg.SvrSslCfg` tool.

Prototype:

```
public class LdapRgyRegistryFactory {
public static RgyRegistry getLdapRgyRegistryInstance
(Properties properties, Map enhancements) throws RgyException;
}
```

If you use `getRgyRegistryInstance()` method that requires authorization, then `propertiesUrl` is referred in the configuration properties file. This `propertiesUrl` must include both `PDAuthorizationContext` and `RgyRegistry` configurations.

Typically, the API uses the `com.tivoli.pd.jcfg.SvrSslCfg` tool to create and manage this combined configuration file. The `PD.jar` file must be accessible by this API and be using the class path.

If API method authorization is not required, specify the `RgyRegistry` configuration properties in the file. The `PD.jar` file is not required when `com.tivoli.pd.rgy.util.RgyConfig` uses this method to create and manage this registry instance.

This method uses *authz.enable-authorization* configuration property to determine whether `PDAuthorizationContext` must be used to create and authorize the API methods.

getLdapRgyRegistryInstance()

Creates an instance of `RgyRegistry` that manipulates LDAP registries. It does not automatically enable authorization or auditing.

If you pass the registry instance to the appropriate registry, you can perform authorization and auditing operations.

These methods are used when the caller wants to provide their own PDAuthorizationContext instance for the RgyRegistry API to authorize its methods.

LdapRgyRegistryFactory.getLdapRgyRegistryInstance(URL propertiesUrl, Map enhancements) closely emulates the LdapRgyRegistry.getRgyRegistryInstance() when combined with

- AuthzRgyRegistryFactory.getRgyRegistryInstance(URL propertiesUrl
- Map enhancements
- RgyRegistry wrappedRgyRegistry
- PDAuthorizationContext pdAuthzContext
- String adminUserId)

Instead of authz.enable-authorization configuration property enabling authorization of the API, supplying a non-null PDAuthorizationContext enables it.

The Administrator user in the authorization of this API is provided as an argument rather than specifying it in the configuration properties file.

Prototype:

```
public class LdapRgyRegistryFactory {
public static RgyRegistry getLdapRgyRegistryInstance
(URL propertiesUrl, Map enhancements) throws RgyException;
public static RgyRegistry getRgyRegistryInstance
(URL propertiesUrl, Map enhancements)throws RgyException;
}
```

com.tivoli.pd.rgy.ldap.AuthzRgyRegistryFactory

This factory creates instances of the RgyRegistry API interface that authorizes and audits other RgyRegistry API instances.

This section explains the following methods in the RgyIterator interface:

- getRgyRegistryInstance()
- getRgyRegistryInstance()
- updateAdminId()
- getPdAuthzContext()

getRgyRegistryInstance()

If authz.enable-authorization is enabled, this version of getRgyRegistryInstance() creates the required PDAuthorizationContext.

The getRgyRegistryInstance() uses authz.pdauthorizatoncontext-user as the administrative user for authorization decisions when it grants access to methods.

The com.tivoli.pd.rgy.ldap.RgyRegistryFactory.getRgyRegistryInstance() instance uses this method for authorization and auditing. The wrappedRgyRegistry is owned by this method, and the instance is automatically closed when appropriate. The caller or calling method must not use or close the instance.

Prototype:

```
public static RgyRegistry getRgyRegistryInstance(URL propertiesUrl,
Map enhancements, RgyRegistry wrappedRgyRegistry) throws RgyException;
```

getRgyRegistryInstance()

Supplies PDAuthorizationContext context, rather than creating it. These methods ignore authz.enable-authorization and authz.pdauthorizatoncontext-user configuration settings and use PDAuthorizationContext and adminUserId. If the PDAuthorizationContext is null, it enables auditing and disables authorization.

Prototype:

```
public static RgyRegistry getRgyRegistryInstance
(Properties properties, Map enhancements, RgyRegistry wrappedRgyRegistry,
 PDAuthorizationContext pdAuthzContext, String adminUserId)
 throws RgyException;
public static RgyRegistry getRgyRegistryInstance
(URL propertiesUrl, Map enhancements, RgyRegistry wrappedRgyRegistry,
 PDAuthorizationContext pdAuthzContext, String adminUserId)
 throws RgyException;
```

updateAdminId()

Updates the administrative user ID used in authorization decisions.

If the specified rgyRegistry instance is not an instance of AuthzRgyRegistry, the method does not perform any action. If authorization is not enabled for AuthzRgyRegistry instance, this method does not perform any action.

Prototype:

```
public static void updateAdminId(RgyRegistry rgyRegistry,
String adminUserId) throws ConfigurationErrorRgyException;
```

getPdAuthzContext()

Returns the PDAuthorizationContext used by the AuthzRgyRegistry instance.

This method returns a null value if:

- The specified rgyRegistry instance is not an instance of AuthzRgyRegistry.
- Authorization is not enabled for the AuthzRgyRegistry instance.

Prototype:

```
public static PDAuthorizationContext getPdAuthzContext
(RgyRegistry rgyRegistry);
```

com.tivoli.pd.rgy.util.RgyConfig

Use com.tivoli.pd.rgy.util.RgyConfig to create and maintain the configuration properties file. You can ignore the PD.jar file or the Security Access Manager Runtime and use this configuration properties file for Registry Direct Java API.

Do not use com.tivoli.pd.rgy.util.RgyConfig to authorize new API methods. Instructions for using this tool are available in the configuration section.

com.tivoli.pd.jcfg.SvrSslCfg

The com.tivoli.pd.jcfg.SvrSslCfg tool supports the combined configuration properties file for PDAuthorizationContext and RgyRegistry.

This tool allows combined use of the Security Access Manager Java API and Registry Direct Java API.

Instructions for using the enhancements to this tool are available in the configuration section.

Old and new API errors

The Registry Direct Java API user, group, and policy administration methods generate errors that closely match the existing user and group administration API errors. This section lists the error codes and explains the errors generated by the API.

Authenticate and changePassword

For the `RgyUser.authenticate()` and `RgyUser.changePassword()`, the Registry Direct Java API generates errors that closely match the existing `azn_util_password_authenticate` and `azn_util_password_change` AZN API errors.

The following table maps the error codes and the API errors:

Table 30. Authentication API error information

RgyException	AZN API Error	AZN status code	AZN API Message
`InvalidCredentialsRgy Exception`	`AZN_S_U_URAF_AUTHEN_FAILED, 0`	*ivacl_s_azn_s_u_uraf_authen_ failed*	*HPDAC1373E* aznAPI User registry authentication failed.
`ServerDownRgyException`	`AZN_S_FAILURE, ivacl_s_registry_server_down`	*ivacl_s_registry_server_down*	*HPDAC0779E* The LDAP registry server is down.
N/A	`AZN_S_FAILURE, ivacl_s_registry_ client_memory_error`	*ivacl_s_registry_client_memory_error*	*HPDAC0777E* LDAP Registry client returned a memory error.
`MultipleDnFoundRgyException InvalidParametersRgyException`	`AZN_S_FAILURE, ivacl_s_registry_client_ bad_ldap_dn`	*ivacl_s_registry_client_bad_ldap_dn*	*HPDAC0772E* The LDAP user registry client returned an error status for the specified DN.
N/A	`AZN_S_FAILURE, ivacl_s_registry_client_ unavailable`	*ivacl_s_registry_client_unavailable*	*HPDAC0771E* The user registry client is unavailable.
(null returned)	`AZN_S_FAILURE, ivauthn_invalid_username`	*vauthn_invalid_username*	*HPDIA0202W* An unknown user name was provided to Access Manager.
`PasswordSetInvalidRgy Exception`	`AZN_S_U_PASSWORD_EXPIRED, 0`	*ivacl_s_azn_s_u_password_expired*	*HPDAC1354E* aznAPI User password expired.
`AccountSetInvalidRgy Exception`	`AZN_S_U_ACCOUNT_DISABLED, 0`	*ivacl_s_azn_s_u_account_disabled*	*HPDAC1364E* aznAPI Account Login is disabled.
`ErrPolicyTodAccessDenied RgyException`	`AZN_S_U_TOD_ACCESS_DENIED, ivauthn_tod_denied`	*ivauthn_tod_denied*	*HPDIA0218W* Authentication by user denied at this time of the day.
`ErrPolicyAcctLockedOutRgy Exception`	`AZN_S_U_ACCOUNT_LOCKEDOUT, 0`	*ivacl_s_azn_s_u_account_ lockedout*	*HPDAC1366E* aznAPI The user account is locked out.
`ErrPolicyPwdTooShortRgy Exception`	`AZN_S_U_PASSWORD_TOO_SHORT, 0`	*ivacl_s_azn_s_u_password_too_short*	*HPDAC1367E* aznAPI New password is too short.
`ErrPolicyPwdHasSpaces RgyException`	`AZN_S_U_PASSWORD_HAS_SPACES, 0`	*ivacl_s_azn_s_u_password_has_spaces*	*HPDAC1368E* aznAPI New password has illegal spaces.
`ErrPolicyPwdTooManyRepeated RgyException`	`AZN_S_U_PASSWORD_TOO _MANY_REPEATED, 0`	*ivacl_s_azn_s_u_password_too_ many_repeated*	*HPDAC1369E* aznAPI New password has too many repeated characters.
`ErrPolicyPwdTooFewAlphaRgy Exception`	`AZN_S_U_PASSWORD_TOO _FEW_ALPHA, 0`	*ivacl_s_azn_s_u_password_ too_few_alpha*	*HPDAC1370E* aznAPI New password has too few alphabetic characters.
`ErrPolicyPwdTooFewNonalpha RgyException`	`AZN_S_U_PASSWORD_TOO _FEW_NONALPHA, 0`	*ivacl_s_azn_s_u_password_too_few_ non_alpha*	*HPDAC1371E* aznAPI New password has too few non-alphabetic characters.

Table 30. Authentication API error information (continued)

RgyException	AZN API Error	AZN status code	AZN API Message
InsufficientAccessRgy Exception	AZN_S_U_INSUFFICIENT _ACCESS, 0	*ivacl_s_azn_s_u_insufficient_access*	*HPDAC1372E* aznAPI Caller does not have the permission to perform requested operation.
ErrPolicyAcctDisabledRgy Exception	AZN_S_U_PASSWORD_ACCT _DISABLED, 0	*ivacl_s_azn_s_u_password_tacct_ disabled*	*HPDAC1374W* aznAPI This account is disabled due to too many failed login attempts.
ErrPolicyAcctLockedOutRgy Exception	AZN_S_U_AUTHEN_FAILED _ACCT_LOCKEDOUT, 0	*ivacl_s_azn_s_u_authen_failed_ acct_lockedout*	*HPDAC1376E* aznAPI User registry authentication failed; the user account has been locked due to too many failed login attempts.
ErrPolicyInvalidAcctDisabled RgyException	AZN_S_U_AUTHEN_FAILED _ACCT_DISABLED, 0	*ivacl_s_azn_s_u_authen_failed_ acct_disabled*	*HPDAC1377E* aznAPI User registry authentication failed; the user account has been disabled due to too many failed login attempts.
N/A	AZN_S_FAILURE, rgy_s_ira_server_in_config_only_ mode	*rgy_s_ira_server_in_config_ only_mode*	*HPDRG0207W* The LDAP server is an IBM Tivoli Directory Server in configuration only mode. Access Manager cannot operate normally with the LDAP server in this mode.
NativePasswordExpiredRgy Exception (when *ldap.enhanced-pwd-policy=true*)	AZN_S_FAILURE, ivauthn_ldap_password_expired (when [ldap] enhanced-pwd-policy = yes)	*ivauthn_ldap_password_expired*	*HPDIA0237W* Authentication failed. The account cannot be logged in because the password expired.
NativePasswordNoModRgyException (when *ldap.enhanced-pwd-policy=true*)	AZN_S_FAILURE, ivauthn_ldap_password_no_mod (when [ldap] enhanced-pwd-policy = yes)	*ivauthn_ldap_password_no_mod*	*HPDIA0318W* The user does not have permission to modify their password.
NativePassword TooYoungRgyException (when *ldap.enhanced-pwd-policy=true*)	AZN_S_FAILURE, ivauthn_ldap_password_ too_young (when [ldap] enhanced-pwd-policy = yes)	*ivauthn_ldap_password_too_young*	*HPDIA0320W* The user cannot change their password until time period elapses after the previous change.
NativePassword InHistoryRgyException (when ldap.enhanced-pwd-policy=true)	AZN_S_FAILURE, ivauthn_ldap_password_ in_history (when [ldap] enhanced-pwd-policy = yes)	*ivauthn_ldap_password_in_history*	*HPDIA0322W* The user is not permitted to use the new password as it was used recently.
NativeAccountLocked RgyException (when ldap.enhanced-pwd-policy=true)	AZN_S_FAILURE, ivauthn_ldap_account_locked	*ivauthn_ldap_account_locked*	*HPDIA0239W* Authentication failed. The account is locked.
NativeAccountInactivated RgyException (when *ldap.enhanced-pwd-policy=true*)	AZN_S_FAILURE, ivauthn_ldap_account_inactivated (when [ldap] enhanced-pwd-policy = yes)	*ivauthn_ldap_account_inactivated*	*HPDIA0241W* Authentication failed. The account is deactivated.
UnhandledRgyException and other RgyExceptions	AZN_S_AZN_S_FAILURE, ivacl_s_registry_client_error	*ivacl_s_registry_client_error*	*HPDAC0773E* The LDAP user registry client returned an unexpected failure status.
WarningPassword ExpiresSoonRgy Exception (when *ldap.enhanced-pwd-policy=true*)	N/A	N/A	N/A

Administration

The Registry Direct Java API user, group, and policy administration methods generate errors that closely match the existing user and group administration API errors.

The following table maps this relationship:

Table 31. Exceptions and the error codes.

RgyException	Error code	Error text
TimeLimitExceededRgyException	*ivmgrd_s_ira2_timelimit_exceeded*	*HPDMG0765W* The request made to the LDAP server exceeded the time limit configured in the server.

Table 31. Exceptions and the error codes. (continued)

RgyException	Error code	Error text
SizeLimitExceededRgyException UnhandledRgyException	*ivmgrd_s_ira2_sizelimit_exceeded*	*HPDMG0766W* The search request exceeded the maximum number of entries the LDAP server can return.
InvalidDnSyntaxRgyException	*ivmgrd_s_ira2_invalid_dn_syntax*	*HPDMG0767E* The Distinguished Name (DN) has an invalid syntax.
InvalidCredentialsRgyException	*ivmgrd_s_ira2_invalid_credentials*	*HPDMG0768E* Unable to log in.
InsufficientAccessRgyException	*ivmgrd_s_ira2_insufficient_access*	*HPDMG0769E* There were insufficient LDAP access privileges for Security Access Manager to create and delete entries in the registry.
ObjectClassViolationRgyException	*ivmgrd_s_ira2_object_class_violation*	*HPDMG0770E* The settings defined for the entry are invalid (object class violation).
ContextNotEmptyRgyException	*ivmgrd_s_ira2_not_allowed_on_nonleaf*	*HPDMG0771E* The settings cannot delete the entry completely because it has unexpected subentries in the LDAP registry. Typically this happens when the deleted user or group is a member of another domain.
AlreadyExistsRgyException	*ivmgrd_s_ira2_already_exists*	*HPDMG0772W* The entry exists.
ServerDownRgyException	*ivmgrd_s_ira2_server_down*	*HPDMG0773E* The request failed because the LDAP server is down.
N/A	*ivmgrd_s_ira2_filter_error*	*HPDMG1052E* A registry memory allocation failed.
N/A	*ivmgrd_s_uraf_no_memory*	*HPDMG0774E* Illegal characters were specified in the LDAP search filter.
N/A	*ivmgrd_s_uraf_no_memory*	*HPDMG1052E* A registry memory allocation failed.
N/A	*ivmgrd_s_ira2_connect_error*	*HPDMG0776E* An error connecting to the LDAP server occurred.
N/A	*ivmgrd_s_ira2_referral_limit_exceeded*	*HPDMG0777W* The LDAP referral limit was exceeded.
N/A	*ivmgrd_s_ira2_ssl_initialize_failed*	*HPDMG0778E* The SSL initialization failed for connection to the LDAP server.
N/A	*ivmgrd_s_ira2_ssl_param_error*	*HPDMG0779E* SSL parameter error occurred when connecting to the LDAP server.
N/A	*ivmgrd_s_ira2_ssl_handshake_failed*	*HPDMG0780E* The SSL handshake failed when connecting to the LDAP server.
N/A	*ivmgrd_s_ira2_ssl_get_cipher_failed*	*HPDMG0781E* SSL failed to establish the requested encryption cipher level when connecting to the LDAP server.
N/A	*ivmgrd_s_ira2_ssl_not_available*	*HPDMG0782E* SSL was not available for connection to the LDAP server
N/A	*ivmgrd_s_ira2_ssl_keyring_not_found*	*HPDMG0783E* The SSL Key Database file was not found for connection to the LDAP server.
N/A	*ivmgrd_s_ira2_ssl_password_not_specified*	*HPDMG0784E* The SSL password was not specified for connection to the LDAP server.
N/A	*ivmgrd_s_uraf_no_memory*	*HPDMG1052E* A registry memory allocation failed.
MultipleDnFoundRgyException	*ivmgrd_s_ira2_multiple_dn_found*	*HPDMG0752E* More than one matching Distinguished Name (DN) was found.

Table 31. Exceptions and the error codes. (continued)

RgyException	Error code	Error text
N/A	*ivmgrd_s_ira2_bad_sec_login_format*	*HPDMG0753E* An invalid format of the authorization mechanism attribute was found in the user entry.
NoSuchAttributeRgyException NoSuchObjectRgyException (null returned)	*ivmgrd_s_ira2_no_entry_found*	*HPDMG0754W* The entry was not found. If creating a user or a group, ensure that the Distinguished Name (DN) specified has the correct syntax and is valid.
N/A	*ivmgrd_s_uraf_group_invalid*	*HPDMG1068E* An invalid group identification or Distinguished Name (DN) was specified.
NoSuchAttributeRgyException	*ivmgrd_s_uraf_member_invalid*	*HPDMG1064E* The group member was not found.
InvalidOldPasswordRgyException	*ivmgrd_s_ira2_invalid_old_password*	*HPDMG0759W* The user name exists in the registry.
IdAlreadyExistsRgyException	*ivmgrd_s_ira2_uid_already_exists*	*HPDMG0756W* Incorrect current password.
IdAlreadyExistsRgyException	*ivmgrd_s_ira2_gid_already_exists*	*HPDMG0760W* The group name exists in the registry.
N/A	*ivmgrd_s_ira2_not_a_user_dn*	*HPDMG0761W* The entry referred to by the Distinguished Name (DN) must be a person entry.
N/A	*ivmgrd_s_ira2_not_a_group_dn*	*HPDMG0762W* The entry referred to by the Distinguished Name (DN) must be a group entry.
N/A	*ivmgrd_s_ira2_ldap_not_supported*	*HPDMG0763E* LDAP is not configured as a registry of users and groups.
AlreadyImportedRgyException	*ivmgrd_s_ira2_entry_already_secuser*	*HPDMG0757W* The Distinguished Name (DN) is already configured as a user.
AlreadyImportedRgyException	*ivmgrd_s_ira2_entry_already_secgroup*	*HPDMG0758W* The Distinguished Name (DN) is already configured as a group.
NativeIdAlreadyExistsRgyException	*ivmgrd_s_ira2_user_already_exists*	*HPDMG0789W* The user Distinguished Name (DN) cannot be created because it exists.
NativeIdAlreadyExistsRgyException	*ivmgrd_s_ira2_group_already_exists*	*HPDMG0790W* The group Distinguished Name (DN) cannot be created because it exists.
ErrInvalidPasswordCharsRgyException	*ivauthn_passwd_policy_violation*	*HPDIA0300W* Password rejected due to policy violation.
ErrPolicyPwdTooShortRgyException	*ivauthn_passwd_too_short*	*HPDIA0301W* Password rejected due to minimum length policy.
ErrPolicyPwdTooFewAlphaRgyException	*ivauthn_passwd_too_few_alphas*	*HPDIA0304W* Password rejected due to the minimum alphabetic characters policy.
ErrPolicyPwdTooFewNonalphaRgy Exception	*ivauthn_passwd_too_few_nonalphas*	*HPDIA0305W* Password rejected due to the minimum non-alphabetic characters policy.
ErrPolicyPwdTooManyRepeatedRgy Exception	*ivauthn_passwd_too_many_repeated*	*HPDIA0303W* Password rejected due to the maximum repeated characters policy.
ErrPolicyPwdHasSpacesRgyException	*ivauthn_passwd_has_spaces*	*HPDIA0302W* Password rejected due to the spaces policy.
NamingViolationRgyException	*ivmgrd_s_ira2_invalid_dn_syntax*	*HPDMG0767E* The Distinguished Name (DN) has an invalid syntax.
TypeOrValueExistsRgyException	*ivmgrd_s_uraf_duplicate_domain_name*	*HPDMG0793E* Duplicate member assignment was attempted. No members have been added.
N/A	*ivmgrd_s_uraf_duplicate_domain_name*	*HPDMG1083W* The domain name exists.

Table 31. Exceptions and the error codes. (continued)

RgyException	Error code	Error text
DomainNotFoundRgyException	*ivmgrd_s_uraf_no_such_domain_name*	*HPDMG1084W* The domain name is unknown.
N/A	*ivmgrd_s_uraf_no_such_domain_name*	*HPDMG1085E* The location specified for creating the management domain does not exist.
N/A	*ivmgrd_s_uraf_no_such_domain_suffix*	*HPDMG1086W* The domain has been created again successfully.
AccountSetInvalidRgyException	*ivmgrd_s_uraf_domain_recreated*	*HPDIA0205W* The user account has expired.
PasswordSetInvalidRgyException	*ivauthn_account_expiredivauthn_password_expired*	*HPDIA0204W* The user password has expired.
N/A	*ivmgrd_s_uraf_domain_name_invalid*	*HPDMG1087E* The domain name specified is invalid.
ErrPolicyAcctDisabledRgyException	*ivauthn_passwd_acct_disabled*	*HPDIA0309W* This account is disabled.
CantChangeDynamicGroupRgyException	*rgy_s_ira_cant_change_dynamic_group*	*HPDRG0200E* The specified group is a dynamic group and cannot be modified.
N/A	*rgy_s_ira_server_in_config_only_mode*	*HPDRG0207W* The LDAP server is an IBM Tivoli Directory Server running in configuration only mode. Security Access Manager does not operate with the LDAP server in this mode.
UnhandledRgyException and other RgyExceptions	*ivmgrd_s_ira2_internal_error*	*HPDMG0764E* An internal error occurred.

Attributes

The API provides access to the Security Access Manager user attributes and group attributes.

The new API provides access to:

- Security Access Manager user attributes, and the native user attributes.
- Security Access Manager group attributes, the description, and *cn* attributes of the native group.

The following table describes the API attribute details:

Table 32. API attribute details

API Constant	Name	Entry	Operation	Description
`MIN_PASSWORD_LENGTH _NAME`	*passwordMinLength*	Security Access Manager User Policy	• Read • Add • Delete • Replace • Create • Import	Minimum length of a password.Multibyte characters are treated as a single character. The value must be a decimal integer. If you do not set this attribute, the API uses the global value.

Table 32. API attribute details (continued)

API Constant	Name	Entry	Operation	Description
PASSWORD_SPACES _NAME	*secPwdSpaces*	Security Access Manager User Policy	• Read • Add • Delete • Replace • Create • Import	Specifies whether to permit space and tabs in passwords. You have 2 choices: • True permits space and tab characters. • False does not permit these characters. If you do not set this attribute, the API uses the global value.
MAX_PASSWORD_REPEATED _CHARS_NAME	*passwordMaxRepeatedChars*	Security Access Manager User Policy	• Read • Add • Delete • Replace • Create • Import	Specifies the maximum number of times a character can be repeated consecutively in a password. The value must be a decimal integer. The value *-1* indicates that there is no limit on the number of times a character can be repeated consecutively. If you do not set this attribute, the API uses the global value.
MIN_PASSWORD_ALPHAS _NAME	*passwordMinAlphaChars*	Security Access Manager User Policy	• Read • Add • Delete • Replace • Create • Import	Specifies the minimum number of alphabetic characters for the password. This set consists of these characters: • UPPERCASE_LETTER: General category `Lu` in the Unicode specification. • LOWERCASE_LETTER: General category `Ll` in the Unicode specification. • TITLECASE_LETTER: General category `Lt` in the Unicode specification. • MODIFIER_LETTER: General category `Lm` in the Unicode specification. • OTHER_LETTER: General category `Lo` in the Unicode specification. Use only decimal integer values. If you do not set this attribute, the API uses the global value.
MIN_PASSWORD_NON_ALPHAS _NAME	*passwordMinOtherChars*	Security Access Manager User Policy	• Read • Add • Delete • Replace • Create • Import	Specifies the minimum number of non-alphabetic characters in the password. This set complements *MIN_PASSWORD_ALPHAS_NAME*. Use only decimal integer values. If you do not set this attribute, the API uses the global value.

Table 32. API attribute details (continued)

API Constant	Name	Entry	Operation	Description
MAX_PASSWORD_AGE _NAME	*passwordMaxAge*	Security Access Manager User Policy	• Read • Add • Delete • Replace • Create • Import	Specifies the number of seconds after the last password change time for which the password is valid. A value 0 (zero) indicates that there is no limit on the maximum number of seconds. If you do not set this attribute, the API uses the global value.
ACCOUNT_EXPIRY_ DATE_NAME	*secAcctExpires*	Security Access Manager User Policy	• Read • Add • Delete • Replace • Create • Import	Specifies time at which the LDAP account expires in Greenwich Median Time. The format is *YYYYMMDDhhmmss.tZ* where: • *YYYY* = year (for example, 2009) • *MM* = month (where January = 01) • *DD* = day of the month (beginning with 01) • *hh* = hour (00 -> 23) • *mm* = minute (00 -> 59) • *ss* = second (00 -> 59) • . = period character • *t* = one tenth of the second (0 -> 9). This is ignored and should be set to 0 • *Z* = this is the 'Z' character. It indicates the time zone is GMT. API recognizes only this format. A special value *unlimited* is accepted and is converted into a value suitable for storage in the underlying registry. **Note:** Upon reading this value, it is not converted into *unlimited*, instead it is the value it was converted to. If you do not set this attribute, the API uses the global value.
DISABLE_TIME _INTERVAL_NAME	*timeExpireLockout*	Security Access Manager User Policy	• Read • Add • Delete • Replace • Create • Import	Specifies the duration in seconds for which the account is locked after *MAX_LOGIN_FAILURES_NAME* login failures have occurred. A value of 0 (zero) disables the account. The value must be a decimal integer >= 0 (zero). If you do not set this attribute, the API uses the global value.

Table 32. API attribute details (continued)

API Constant	Name	Entry	Operation	Description
MAX_LOGIN _FAILURES_NAME		Security Access Manager User Policy	• Read • Add • Delete • Replace • Create • Import	Specifies the number of login failures that can occur before the software lock or disables the account. Disabling or the time period for the lock out depends on DISABLE_TIME_INTERVAL_NAME. The value must be a decimal integer >=0 (zero). See the ldap.login-failure-persistent and ldap.late-lockout-notification configuration options. If you do not set this attribute, the API uses the global value.

Table 32. API attribute details (continued)

API Constant	Name	Entry	Operation	Description
TOD_ACCESS_NAME	*maxFailedLogins*	Security Access Manager User Policy	• Read • Add • Delete • Replace • Create • Import	Limits authentication to particular days of the week and a specific range of time during the day. The format of the policy is *days:start:end:zone* where: • *days* - a decimal integer that represents a bit mask of days of the week. – SUNDAY=1 – MONDAY=2 – TUESDAY=4 – WEDNESDAY=8 – THURSDAY=16 – FRIDAY=32 – SATURDAY=64 • *start* - the decimal integer that represents the starting minute of the day of allowed access. • *end* - a decimal integer that represents the ending minute of the day of allowed access. • *zone* - a decimal integer that, when set to 1, indicates that GMT must be used to determine the current time of day and the day of the week against which to evaluate this policy. If you set any other value, the local default time zone is used. If you do not set this attribute, the API uses the global value. **Note:** When you set a password policy, you provide a list of days, start time, and end time. The start time and end time apply to each day on the list. If the specified start time is later than the specified end time, then the access is allowed until the specified end time is reached the next day.

Table 32. API attribute details (continued)

API Constant	Name	Entry	Operation	Description
MAX_CONCURRENT _WEB_SESSIONS_NAME	*secTODAccess*F	Security Access Manager User Policy	• Read • Add • Delete • Replace • Create • Import	The maximum number of concurrent web login for the user. This API does not use this value directly, but other applications use this value. The value must be a valid decimal integer. There are special negative values, which are: • *-3* When set, a new login displaces (logout) other login sessions of the same user. • *-4* When set, the number of concurrent logins are not limited. If you do not set this attribute, the API uses the global value.
SEC_ACCT_VALID_NAME	*secAcctValid*	Security Access Manager User	• Read • Replace • Create • Import	Indicates the account validity status.The permitted values are *true* and *false*. When set to *false*, you cannot log in to an account.
SEC_PWD_VALID_NAME	*secPwdValid*	Security Access Manager User	• Read • Replace • Create • Import	Indicates the password validity setting.This attribute can be set only to *true* and *false*. When set to *false*, the user must change the password at next logon.
SEC_DN_NAME	*secDN*	Security Access Manager User	• Read	Internal use only. Use `getNativeId()` instead of `SEC_DN_NAME`.
SEC_UUID_NAME	*secUUID*	Security Access Manager User	• Read • Create • Import	Specifies the Universally Unique ID. This attribute is normally generated by the API for the user. It is mostly used by the Authorization API when verifying ACLs. You can supply this value when you create or import a user. You cannot modify this value after you set it. Do not specify any value for this parameter except when you recover accounts that were accidentally deleted.
SEC_LOGIN_TYPE _NAME	*secLoginType*	Security Access Manager User	• Read	Internal use only.
SEC_CERT_DN_NAME	*secCertDN*	Security Access Manager User	• Read	Internal use only.
SEC_CERT_SERIAL _NUMBER_NAME	*secCertSerialNumber*	Security Access Manager User	• Read	Internal use only.
SEC_HAS_POLICY_NAME	*secHasPolicy*	Security Access Manager User	• Read	Internal use only.
SEC_AUTHORITY_NAME	*secAuthority*	Security Access Manager User	• Read	Internal use only.

Table 32. API attribute details (continued)

API Constant	Name	Entry	Operation	Description
PRINCIPAL_NAME_NAME	*principalName*	Security Access Manager User	• Read	Internal use only. Use getId() instead of this attribute.
SEC_PWD_FAILURES_NAME	*secPwdFailures*	Security Access Manager User Policy State	• Read	Internal use only. Specifies the number of consecutive authentication failures because of wrong password. This policy is a mechanism to enforce the MAX_LOGIN_FAILURES_NAME policy only if the *ldap.login-failures-persistent* option is enabled.
SEC_PWD_LAST_CHANGED _NAME	*secPwdLastChanged*	Security Access Manager User Policy State	• Read	Specifies the time when the password was last changed. This policy is a mechanism to enforce the MAX_PASSWORD_AGE_NAME policy. The value is updated to the current date when SEC_PWD_VALID_NAME is set to true.
SEC_PWD_LAST_USED _NAME	*secPwdLastUsed*	Security Access Manager User Policy State	• Read	Specifies the last time the that user logged in. This value is updated every time Security Access Manager successfully authenticates a user. This value is updated only for password-based authentication. The option *ldap.enable-last-login* is set to true.
SEC_DOMAIN_ID_NAME	*secDomainId*	Security Access Manager User	• Read	Internal use only.
SEC_PWD_LAST_FAILED _NAME	*secPwdLastFailed*	Security Access Manager User Policy State	• Read	Internal use only. Records the time of the last failed login to authenticate with the correct password. This value is a part of the mechanism to enforce the DISABLE_TIME_INTERVAL_NAME policy. **Note:** Some operations might be restricted by the LDAP.
SEC_PWD_UNLOCK_TIME _NAME	*secPwdUnlockTime*	Security Access Manager User Policy State	• Read	Internal use only. Records the duration for which the account is locked. This value is a part of the mechanism to enforce the DISABLE_TIME_INTERVAL_NAME policy.
COMMON_NAME_NAME	*cn*	Native User and Native Group	• Read • Add • Delete • Replace • Create • Import	Required when you create users or groups. **Note:** LDAP server might restrict some operations.

Table 32. API attribute details (continued)

API Constant	Name	Entry	Operation	Description
SURNAME_NAME	*sn*	Native User	• Read • Add • Delete • Replace • Create • Import	Required when you create users. **Note:** LDAP server might restrict some operations.
UID_NAME	*uid*	Native User	• Read • Add • Delete • Replace • Create • Import	Specifies the LDAP Unique ID attribute name. This attribute is an optional attribute when you create a RgyUser. If you do not specify a value, this parameter is set to the userId or uid value in the leading RDN® of the userNativeId. **Note:** LDAP server might restrict some operations.
OBJECT_CLASS_NAME	*objectClass*	Native User and Native Group	• Read • Create	Internal use only. Indicates the LDAP object class attribute name. This attribute contains the native LDAP objectClass values for the native entry.
DESCRIPTION_NAME	*description*	Native User and Native Group	• Read • Add • Delete • Replace • Create • Import	Indicates the LDAP description attribute name. Optional attribute when creating a new RgyUser or RgyGroup. **Note:** LDAP server might restrict some operations.
IS_SEC_ENTITY_NAME	*isSecEntity*	Security Access Manager User and Security Access Manager Group	• Read	Set to *true* if the account is a Security Access Manager enabled account. This attribute is virtual, and is dynamically determines instead of being stored in the LDAP registry.
IS_GSO_USER_NAME	*isGSOUser*	Security Access Manager User	• Read	Set to *true* if the account is a Global Sign-On (web SSO) enabled account. This attribute is virtual, and is dynamically determines instead of being stored in the LDAP registry.
*	*	Native User	• Read • Add • Delete • Replace • Create • Import	Indicates a native user entry that might have additional attributes for the user. If the LDAP server permits, the values are updated or deleted. **Note:** LDAP servers might restrict some operations.

Table 32. API attribute details (continued)

API Constant	Name	Entry	Operation	Description
RESOURCE_CREDENTIALS_NAME	*resourceCredentials*	Security Access Manager User	• Read	If the account is a global sign on-enabled and has resource credentials created for it, then this attribute will contain the resource credentials of the user. This is a virtual attribute that is not stored directly in the LDAP registry. Rather, it is dynamically determined from multiple entry attributes in LDAP. Each value for the attribute represents one resource credential and has the resources credential values condensed into one string. The API provide methods to expand these resource credential values into separate strings.

Error and trace logging

Error and trace logging use the Java logging mechanism. The Java logger names are: *com.tivoli.pd.rgy.authz* and *com.tivoli.pd.rgy.ldap*. For basic Java Runtime installation, the configuration of the logger output is in the `lib/logging.properties` file. Graphic user interfaces (GUI) are available for environments such as WebSphere to configure and enable various log levels. The trace level `CONFIG` is also available for tracing configuration options.

Basic JRE example output

If the `lib/logging.properties` file is set to use `java.util.logging.XMLFormatter` for debug logging, the output appears as follows:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE log SYSTEM "logger.dtd">
<log>
<record>
<date>2009-11-16T16:28:35</date>
<millis>1258417715058</millis>
<sequence>0</sequence>
<logger>com.tivoli.pd.rgy.ldap</logger>
<level>FINER</level>
<class>com.tivoli.pd.rgy.ldap.LdapRgyRegistry</class>
<method>LdapRgyRegistry</method>
<thread>10</thread>
<message>RETURN</message>
</record>
<record>
<date>2009-11-16T16:28:35</date>
<millis>1258417715136</millis>
<sequence>1</sequence>
<logger>com.tivoli.pd.rgy.ldap</logger>
<level>FINER</level>
<class>com.tivoli.pd.rgy.ldap.LdapRgyRegistry</class>
<method>configure</method>
<thread>10</thread>
<message>ENTRY</message>
</record>
```

Auditing

The Registry Direct API provides a set of Java interfaces. The code used for manipulating the LDAP (the equivalent of the Security Access Manager IRA C code) implements these interfaces.

The auditing feature does not support Common Audit and Reporting Services (CARS). The auditing feature supports Security Access Manager XML file auditing.

Java logger behavior

This section describes how the auditing feature behaves.

The auditing feature uses a Java logger framework, but directs the output to a configurable file. The name of the audit Java logger is visible on the Java logger interfaces.

You can enable or disable the output to the logger and adjust the level of log output by varying the log level in the Java run time.

The audit code uses a custom formatter. This means that the output format is the Security Access Manager XML audit file format. The Java logger namespace for auditing has a configurable component that allows each Java API exploiter to have a separate audit file and namespace.

The following are the loggers for authentication and management auditing:

```
com.tivoli.pd.rgy.audit.{blade}.authn - Authentication audit logging
com.tivoli.pd.rgy.audit.{blade}.mgmt - Management audit logging
```

where {*blade*} is the value of the configuration property *appsvr-servername*.

Note: If there are two separate instances of `RgyRegistry` in the same JVM, that use the same *appsvr-servername* value, the instance shares one audit log file. The instance that was created first defines the audit log file to be used. The second instance continues to use any existing logger of the same name.

File format

The Policy Server output is emulated by the Registry Direct Java API.

Information that is unavailable is either hardcoded to a pre-set value or is excluded.

Start and stop events:

The two audit components `mgmt` and `authn` mimic the output of the audit components of the same name in the Security Access Manager Policy Server.

These components generate:

- an audit start event when the `RgyRegistry` instance is generated.
- an audit stop event when the `RgyRegistry` instance is closed.

Sample start event for `mgmt` component is as follows:

```
<event rev="1.2">
 <date>2009-11-19-10:34:45.380-08:00I-----</date>
 <outcome status="0">0</outcome>
 <originator blade="testapp-tam611">
  <component rev="1.1">mgmt</component>
```

```
 <event_id>117</event_id>
 <action>0</action>
 <location>localhost</location>
</originator>
<target resource="5">
 <object></object>
</target>
<data>
 <audit event="Start"/>
</data>
</event>
```

Sample start event for `authn` component is as follows:

```
<event rev="1.2">
 <date>2009-11-19-10:34:45.492-08:00I-----</date>
 <outcome status="0">0</outcome>
 <originator blade="testapp-tam611">
  <component rev="1.1">authn</component>
  <event_id>117</event_id>
  <action>0</action>
  <location>localhost</location>
 </originator>
 <target resource="7">
  <object></object>
 </target>
 <data>
  <audit event="Start"/>
 </data>
</event>
```

Management events:

This section provides a sample output of a management operation.

A sample output of a management operation such as `RgyRegistry.authenticate()` is:

```
<event rev="1.2">
 <date>2009-11-19-10:34:45.580-08:00I-----</date>
 <outcome status="0">0</outcome>
 <originator blade="testapp-tam611">
  <component rev="1.1">mgmt</component>
  <event_id>13401</event_id>
  <action>13401</action>
  <location>tam611.ibm.com</location>
 </originator>
 <accessor name="">
  <principal auth="IV_UNAUTH_V3.0" domain="Default">unauthenticated</principal>
 </accessor>
 <target resource="5">
  <object></object>
 </target>
 <mgmtinfo>
  <command>USER CREATE</command>
  <objname>testuser0</objname>
  <objtype>user</objtype>
  <objname_rgy>cn=testuser0,o=ibm,c=us</objname_rgy
  <parm><name>gsouser</name><value>false</value></parm>
  <parm><name>nopwdpolicy</name><value>false</value></parm>
  <parm><name>dn</name><value>cn=testuser0,o=ibm,c=us</value></parm>
  <parm><name>loginid</name><value>testuser0</value></parm>
  <parm><name>accountvalid</name><value>TRUE</value></parm>
  <parm><name>maxloginfailures</name><value>2</value></parm>
  <parm><name>cn</name><value>testuser0</value></parm>
  <parm><name>minpasswordlength</name><value>8</value></parm>
  <parm><name>accountexpirydate</name><value>20091119193445.0Z</value></parm>
```

```
  <parm><name>sn</name><value>user0</value></parm>
  <parm><name>disabletimeinterval</name><value>0</value></parm>
 </mgmtinfo>
 <data></data>
</event>
```

The preceding sample demonstrates how to provide additional attributes to the existing API. The attribute names are mapped, wherever possible, to existing API audit attribute names.

You can create this output using `createUser()`:

```
registry = LdapRgyRegistryFactory.getRgyRegistryInstance(propertiesUrl, null);
String sn = "user0";
String id = "test"+sn;
String nativeId = "cn="+id+",o=ibm,c=us";
RgyAttributes rgyAttributes = registry.newRgyAttributes();
rgyAttributes.putAttribute(RgyAttributes.COMMON_NAME_NAME, id);
rgyAttributes.putAttribute(RgyAttributes.SURNAME_NAME, sn);
rgyAttributes.putAttribute(RgyAttributes.MIN_PASSWORD_LENGTH_NAME, "8");
rgyAttributes.putAttribute(RgyAttributes.MAX_LOGIN_FAILURES_NAME, "2");
rgyAttributes.putAttribute(RgyAttributes.DISABLE_TIME_INTERVAL_NAME, "0");
Date currentTime = new Date();
currentTime.setTime(currentTime.getTime() + (3600 * 1000L));
String registryDate = null;
registryDate = registry.toRegistryDate(currentTime);
rgyAttributes.putAttribute(RgyAttributes.ACCOUNT_EXPIRY_DATE_NAME, registryDate);
rgyAttributes.putAttribute(RgyAttributes.SEC_ACCT_VALID_NAME,
RgyAttributes.BOOL_TRUE_VALUE);
RgyUser user = registry.createUser(null, id, nativeId,
"passw0rd".toCharArray(), false, rgyAttributes, null);
```

Authentication events:

This section provides a sample output of an authentication operation.

Authentication methods such as `RgyUser.authenticate()` produces output as follows:

```
<event rev="1.2">
<date>2009-11-19-10:34:49.175-08:00I-----</date>
<outcome status="1" exception="com.tivoli.pd.rgy.exception.
  InvalidCredentialsRgyException">1</outcome>
 <originator blade="testapp-tam611">
  <component rev="1.1">authn</component>
  <event_id>101</event_id>
  <action>0</action>
  <location>tam611.ibm.com</location>
 </originator>
 <accessor name="">
  <principal auth="IV_LDAP_V3.0" domain="Default">testuser0</principal>
  <name_in_rgy>cn=testuser0,o=ibm,c=us</name_in_rgy>
 </accessor>
 <target resource="7">
  <object></object>
 </target>
 <authntype>formsPassword</authntype>
 <data>Password Failure: testuser0</data>
</event>
```

The preceding example is generated using the `authenticate()` method as shown in the following sample code:

```
testuser = registry.getUser(null, "testuser0");
testuser.authenticate("passw1rd".toCharArray());
```

Java logger namespace:

This section describes the Java logger namespace.

The Java logger namespaces and their descriptions are provided in the following table:

Table 33. Java logger namespace

Package	Description
`com.tivoli.pd.rgy.ldap`	LDAP `LdapRgyRegistry` trace and logging
`com.tivoli.pd.rgy.authz`	Authorization and audit `AuthzRgyRegistry` trace and logging
`com.tivoli.pd.rgy.audit.{blade}.authn`	Authentication audit logging
`com.tivoli.pd.rgy.audit.{blade}.mgmt`	Management audit logging

Authorization

This section describes the API authorization configuration.

The API authorization cannot fully emulate the authorization performed by the Policy Server (pdmgrd). The group and user delegation cannot be emulated and the API assumes that the functionality is not being used.

You can configure the Registry Direct Java API to not support delegated user and group administration. By default, the API does not support ACLs placed on child objects of */Management/Groups*. Unlike the Security Access Manager Java API, the Registry Direct Java API does not create and delete protected objects under */Management/Groups*. But the registry API affects authorization operations.

No ACLs are placed on child objects of */Management/Groups*.

Authorization permission checks

The table in this section describes administrative operations and corresponding authorization permissions.

The information in the table assumes that:

- No ACLs are added on delegation protected objects under */Management/Groups* without taking advantage of delegated user and group management.
- All delegation protected objects under */Management/Groups* inherit the same ACL that */Management/Groups* uses.

  You do not have to verify whether each group has protected objects that are associated with it. Verifying */Management/Groups* is sufficient.
- `adminid` as the account that is requesting administration operation.

Table 34. Authorization permissions for groups

Administrative operations	Permission	Verification of permissions
`RgyRegistry.createUser({groups})`	*"N"*	If no group exists, check permission on */Management/Users*. Otherwise, check permission on */Management/Groups*.
`RgyUser.importNativeUser({group})`	*"N"*	If the specified group `{group}` does not exist check the permission on */Management/Users*. Otherwise, check the permission on */Management/Groups*.

Table 34. Authorization permissions for groups (continued)

Administrative operations	Permission	Verification of permissions

`RgyUser.attributeAdd()`
`RgyUser.attributeDelete()`
 	
`RgyUser.AttributeReplace()` for user `{userid}` for all attributes except: `secAcctValid secPwdValid` all policy attributes.
	"m"	Check permission on */Management/Groups* if not permitted. If the user does not have group membership, deny access and check permission on */Management/Groups*.

`RgyUser.attributeAdd()`
`RgyUser.attributeDelete()`
 	
`RgyUser.AttributeReplace()` for user `{userid}` for attributes: `secAcctValid secPwdValid`
	"m"	If the `{userid}` is the same as `{adminid}` deny access. Otherwise, check permission on */Management/Users*. If `{userid}` has no group membership, deny access. Otherwise, check permission on */Management/Groups*.

`RgyUser.setPassword()` for user `{userid}`	*"W"*	If the `{userid}` is the same as `{adminid}` permit access. Otherwise, check permissions on */Management/Users*. If `{userid}` has no group membership, deny access. Otherwise, check permission on */Management/Groups*.
`RgyRegistry.deleteUser({userid})`	*"d"*	If the `{userid}` is the same as `{adminid}` deny access. Otherwise, check permissions on */Management/Users*. If `{userid}` has no group membership, deny access. otherwise, check permission on */Management/Groups*.
`RgyUser.listGroups()` `RgyUser.listNativeGroups()`		The permission check that is performed for `RgyRegistry.getUser()` was sufficient. Permit access.
`RgyRegistry.getUser({userid})`	*"v"*	If the `{userid}` is the same as `{adminid}` permit access. Otherwise, check permission on */Management/Users*. If `{userid}` does not belong to a group, deny permission. Otherwise, check permission on */Management/Users*.
`RgyRegistry.getNativeUser({dn})`	*"v"*	Map the distinguished name `{dn}` to the group `{groupid}`. If mapping does not exists, permit. If a mapping exists, then check permission on */Management/Users*. If `{userid}` does not have any group, check permission on the */Management/Users*.
`RgyUser.getAttributeValues()` `RgyUser.getOneAttributeValue()` for all non-policy attributes.		The permission check done for `RgyRegistry.getUser()` was sufficient. Permit access.
`RgyRegistry.listUsers()` `RgyRegistry.listNativeUsers()`		There are delegation effects on the operation results in the follow-up table.
`RgyRegistry.createGroup()`	*"N"*	Check permission on */Management/Groups*.
`RgyRegistry.deleteGroup()`	*"d"*	Check permission on */Management/Groups*.
`RgyGroup.importNativeGroup()`	*"N"*	Check permission on */Management/Groups*.
`RgyRegistry.listGroups()` `RgyRegistry.listNativeGroups()`		Permit (there are delegation effects on the operation results in the follow-up table).
`RgyGroup.addMembers()`	*"A"*	Check permission on */Management/Groups*.
`group modify {groupid} description`	*"m"*	Check permission on */Management/Groups*.
`RgyGroup.removeMembers()`	*"A"*	Check permission on */Management/Groups*.
`RgyRegistry.getGroup()`	*"v"*	Check permission on */Management/Groups*.
`RgyRegistry.getNativeGroup({dn})`	*"v"*	Map the distinguished name {dn} to the group {groupid}. If no mapping exists, permit. Otherwise, check permission on */Management/Groups*.

Table 34. Authorization permissions for groups (continued)

Administrative operations	Permission	Verification of permissions

`RgyGroup.listMemberIds()`
 	
`RgyGroup.listMemberNativeIds()`
	"v"	Check permission on */Management/Groups*.

`RgyUser.getAttributeValues()` `RgyUser.getOneAttributeValue()` for all policy attributes.		This differs from the permission check performed by the regular Security Access Manager API. The regular Security Access Manager API checks for the permission *IVACTION_VIEW* "v" on */Management/Policy*. This API does not perform any additional permission checks. It completes permission check when fetching the user using `RgyRegistry.getUser()` method.
`RgyUser.attributeAdd()RgyUser.attributeDelete()` `RgyUser.AttributeReplace()` for all policy attributes.	*"m"*	Check permission on */Management/Groups*.

Unlike `pdadmin` or the Security Access Manager Java management API, when you use the Registry Direct Java API, you must fetch the user before you modify the user. Assign view permission (*v*) and modify permission (*m*) for the administrator to modify either a user or a group.

Residual effects of delegated administration on admin results

If operations are permitted, additional permissions can be verified by the API to determine if a different subset of result must be returned. The permission check in this case does not permit or deny the whole operation. It affects only the result set returned, instead.

The following table shows the additional effects of delegated administration on the result set. It also assumes that no ACLs are added on delegation protected objects under */Management/Groups*. This assumption reduces many of the delegated administration complexities into simpler behavior.

Table 35. List of operations and permissions to be checked

Operation	Permission to be checked
`RgyUser.listGroups() RgyUser.listNativeGroups()`	Check permission `DELADMIN_VIEW("v") on "/Management/Users"`. If permitted return `group list`else check permission `DELADMIN_VIEW on "/Management/Groups"`. If permitted return `group list` else return an empty list.
`RgyRegistry.listUsers({pattern})`	Check permission `DELADMIN_VIEW("v")on "/Management/Users"`. If permitted, return list of users matching `{pattern}`else check permission `DELADMIN_VIEW("v") on "/Management/Groups"`. If permitted, return list of users matching the`pattern`. In this case, the current API returns only users that are a member of at least one Security Access Manager group, this API will not enforce this restriction else return an empty list.

Table 35. List of operations and permissions to be checked (continued)

Operation	Permission to be checked
RgyRegistry.listNativeUsers({pattern})	Check permission DELADMIN_VIEW("v") on "/Management/Users". If permitted, return list of users dns with attribute matching pattern (both Security Access Manager and non-Security Access Manager user dn) else check permission DELADMIN_VIEW("v") on "/Management/Groups". If permitted, return list of user dn with attribute matching pattern. In this case, the current API only returns dn where the actual dn matches the pattern and only dn of Security Access Manager users, that are a member of at least one Security Access Manager group else return empty list.
RgyRegistry.createGroup()	New code is unable to create group delegation protected object.
RgyGroup.importNativeGroup()	New code is unable to create group delegation protected object.
RgyRegistry.deleteGroup()	New code is unable to delete group delegation protected object, if it exists.
RgyRegistry.listGroups({pattern})	List all groups with Security Access Manager ID matching pattern. Check configuration option "[delegated-admin]authorize-group-list = yes/no" if authorize-group-listreturn groupList. Otherwise, check perm DELADMIN_VIEW("v") on "/Management/Groups". If permitted, return the groupList. Otherwise, return empty list.
RgyRegistry.listNativeGroups({pattern})	groupList = list all group DNs with attribute matching {pattern} check config option "[delegated-admin] authorize-group-list = yes/no" if !authorize-group-list: return groupList else check permisson DELADMIN_VIEW("v") on "/Management/Groups" if permitted: return groupList else return empty list

API Specifications

See the Javadoc for packages under com.tivoli.pd.rgy in the product DVD.

Installation and configuration

You can install, configure, and upgrade the Registry Direct API by following these instructions.

Upgrade

Use the PD.jar available with the Registry Direct Java API Reference to configure the features. The WebSphere Application Server installation program includes PD.jar in the JRE class path. Update the class path manually or avoid references to this PD.jar when configuring the Registry Direct API along with the Security Access Manager Java API. After configuration, you can use the Registry Direct API along with the earlier versions of PD.jar.

Installation and packaging

The Registry Direct API PD.jar package includes the PDjrte package, which is installed during the API installation.

The exploiter of the API must include the Registry Direct API PD.jar JAR file in their application class path.

The Registry Direct API JAR file is com.tivoli.pd.rgy.jar. This file is installed in the PolicyDirector/java/export/rgy directory.

Configuration

There are two configuration options for configuring the Registry Direct API and the Security Access Manager Java API.

Two configuration options are available:

- **Standalone usage** - the configuration permits stand-alone usage of the Registry Direct API.
- **Combined usage** - the configuration allows convenient and combined use of both Security Access Manager Java API and the Registry Direct API.

Using both Security Access Manager Java API and the Registry Direct API

This section describes how to use SvrSslCfg to configure the Security Access Manager Java API and the Registry Direct API at the same time.

The SvrSslCfg that is provided in PD.jar is enhanced to permit simultaneous configuration of the current Java API and Registry Direct Java API.

Both configurations share the same LDAP identity and configuration properties file. This feature is useful if authorization is enabled in the new API, which requires the current API to provide the identity and configuration properties file.

If SvrSslCfg has *–ldap_mgmt* set to *true*, it creates a Security Access Manager identity for the application, the identity is also added to the Security Access Manager group, *SecurityGroup*. This group membership gives the underlying LDAP identity the required privileges to administer user and groups in LDAP.SvrSslCfg generates a random password, if a password is not supplied, for the identity. It obfuscates and stores the password in the configuration file. The underlying LDAP identity and password that are stored in the configuration file is used by the Registry Direct API to bind to LDAP to perform administration and authentication.

Stand-alone configuration

This section describes the stand-alone configuration.

To use the *authz.enable-authorization* option in the Registry Direct API, a PDAuthorizationContext instance from the Security Access Manager Java API must be used. In such a case, it is better to use SvrSslCfg.

In stand-alone configuration, the LDAP identity that is used to access LDAP and perform the administration updates must be manually created. A simple method is to use access manager to create the LDAP identity. For example:

```
pdadmin sec_master> user create -no-password-policy
testapi cn=testapi,o=ibm,c=us testapi api passw0rd
( SecurityGroup ivacld-servers remote-acl-users )
```

After you create the Security Access Manager identity, you do not need the Security Access Manager identity. If required, you can delete the Security Access Manager identity by using the following command:

```
pdadmin sec_master> user delete testapi
```

The -registry option is not specified. As a result, the underlying LDAP account is left along with its group memberships. You do not need to remove the Security

Access Manager identity from the LDAP account. Leaving the Security Access Manager identity associated with the LDAP account makes it easier to manage the account.

Note: If native LDAP password policy (not Security Access Manager Policy) is enabled and affects this LDAP account (for example, Global policy), it must not reset the account, or account password or set them to expire state. To avoid this apply native LDAP policy.

The configuration tool `RgyConfig` is provided in the JAR file along with the new API. The usage is as follows:

Usage:

```
   java com.tivoli.pd.rgy.util.RgyConfig <file> <command> [options]
   <file> configuration properties file path name
   <command> is one of:
create <mgmt_domain> <local_domain> <ldap.svrs>
    <ldap.bind_dn> <ldap.bind_pwd> [<ldap.ssl_truststore>
    <ldap.ssl_truststore_pwd>
    load <input properties file>
set <name> <value>
remove <name>
get <name>
list
```

The usage for a non-SSL example is:

```
java com.tivoli.pd.rgy.util.RgyConfig /tmp/testapi.properties create Default Default
"ldaphostname:389:readwrite:5" "cn=testapi,o=ibm,c=us" passw0rd
```

After you create the properties file, you can manipulate the additional properties. The example to set `ldap.enable-last-login` property is as follows:

```
java com.tivoli.pd.rgy.util.RgyConfig /tmp/testapi.properties set ldap.
enable-last-login true
```

When you use the `RgyConfig` tool, you must manually create the server identity. Ensure that the Security Access Manager subdomains include the server identity in the `remote-acl-users` group of Security Access Manager management domain. If Security Access Manager domain is not the default domain, the following additional steps are needed.

1. Create a file `groupmodify.ldif` with the following contents:

   ```
   dn: cn=remote-acl-users,cn=SecurityGroups,secAuthority=Default
   changetype: modify
   add: member
   member: cn=testapp/tam611,cn=SecurityDaemons,secAuthority=testdom,
   cn=Subdomains,secAuthority=Default
   ```

   Where `member` is the LDAP DN of your application. This value is provided as `ldap.bind_dn`argument to `java com.tivoli.pd.rgy.util.RgyConfig`. Alternately, you can determine this value from `ldap.bind-dn` stored in the generated properties file.
2. Update LDAP by using `ldapmodify` command.

   ```
   ldapmodify -p 389 -h localhost -D "cn=root" -w passw0rd -f groupmodify.ldif
   ```

   Replace `localhost` and `passw0rd` with values appropriate for your setup.

Note: You do not have to install or configure Security Access Manager to use the new API in stand-alone mode.

Configuration options

The following table describes the configuration options for the Security Access Manager Java API and the Registry Direct API.

Table 36. Configuration options

Java Option Name	Existing Comparable Option	Existence in current Java Config	Default	Valid Range	Description
`ldap.mgmt`	`[ldap] enabled`	New, Optional	false	*true, false*	Set this option `true` to enable LDAP management.
`mgmt_domain`	`[manager] management-domain`	Already Present, Required		*valid domain string*	Security Access Manager Management Domain name. Required to determine the location of subdomain in the registry. Sub domains are located relative to the Management Domain LDAP location.
`local_domain`	`[ssl] ssl-local-domain`	Already Present, Optional		*valid domain string*	The name of the default domain that is used when the Management API does not provide a domain name. If you do not provide a value, the value from `mgmt_domain` configuration option is used.
`ldap.dynamic-groups-enabled`	`[ldap] dynamic-groups-enabled`	New, Optional	false	*true, false*	Enables support of dynamic groups for some LDAP server types by using the `memberURL` attribute. Security Access Manager supports dynamic groups with Tivoli Directory Server regardless of this setting. This stanza entry is supported for Oracle System Directory Server.
`ldap.enable-last-login`	`[ldap]enable-last-login`	New, Optional		*true, false*	Sets an option to store the last login date in LDAP each login.
`ldap.mgmt-domain-suffix`	`[ldap] secauthority-suffix`	New, Optional	Will be automatic-ally located.	*valid LDAP suffix string*	Specify the valid LDAP suffix string for the Domain Management of the Security Access Manager.
`ldap.ignore-suffix`	`[ldap] ignore-suffix`	New, Optional	Empty list	*list of valid LDAP suffix strings*	Ignore LDAP server suffix when searching for user and group information. Suffixes `cn=localhost` , `cn=pwdpolicy`, `cn=configuration`, and the suffixes that are specified in the `subschemasubentry` and `changelog` values are always ignored. `SvrSslCfg` accepts multiple values by using ",," (double comma) separator. The configuration file uses ";" (semicolons) internally as a separator.
`ldap.max-server-connections`	`[ldap] max-server-connections`	New, Optional	16	*2 -> 4096*	Indicates the maximum number of connections that can exist to the LDAP server.
`ldap.user-objectclass`	`[ldap] user-objectclass`	New, Optional		*Defaults vary depending on LDAP server type*	When provided to the configuration tool, it contains a list of comma-separated object class names to set when creating a native user entry in LDAP. For example: *top,person, organizationalPerson, inetOrgPerson,ePerson*. `SvrSslCfg` that modifies the list to be ";" (semicolon) separated when it places it in the configuration properties file.
`ldap.static-group-objectclass`	`[ldap] static-group-objectclass`	New, Optional		*Defaults vary depending on LDAP server type*	When provided to the configuration tool, it contains a list of comma-separated `objectClass` names to set when creating a native group entry in LDAP. Only non-dynamic groups are created by Security Access Manager. For example, *top,groupOfNames*. `SvrSslCfg` modifies the list to be ';' (semicolon) separated when it places it in the configuration properties file.

Table 36. Configuration options (continued)

Java Option Name	Existing Comparable Option	Existence in current Java Config	Default	Valid Range	Description
`ldap.user-search-filter`	`[ldap] user-search-filter`	New, Optional	Defaults vary depending on LDAP server type.	*valid LDAP search filter string*	An LDAP search filter that selects any native user entry. For example: *(|(objectclass=ePerson) (objectclass=Person))*.
`ldap.group-search-filter`	`[ldap] group-search-filter`	New, Optional	Defaults vary depending on LDAP server type.	*valid LDAP search filter string*	An LDAP search filter that selects any native group entry. For example: *(|(objectclass=accessGroup) (objectclass=groupOfNames) (objectclass=groupOfUniqueNames) (objectclass=groupOfURLs))*
`ldap.svrs`	`[ldap] host, port, ssl-port, and replica`	New, Required		*valid host string, port 1 -> 65535, type readwrite or readonly, pref 0 -> 10*	A comma-separated list of LDAP server details. Each server detail is a colon separated set of attributes of the form: `host:port:type:rank[,host2: port2:type2:rank2[,...]]` where `type` is either `readwrite` or `readonly` and rank is a value from 0 to 10. For example: `ldaphost:389:readwrite:5` is modified to a list of LDAP server details that are separated by ';'s.
`ldap.ssl-enable`	`[ldap] ssl-enable`	New, Optional	False	*true, false*	Set this option to true to enable SSL to the LDAP server.
`ldap.fips`	`[ssl] ssl-enable-fips`	New, Optional	False	*true, false*	Deprecated: replaced by `ldap.compliance`. Use `ldap.compliance=fips` for `ldap.fips=true`. Use `ldap.compliance=none` for `ldap.fips=false`. Set this option to *true* to use FIPS mode with the TLS connections to the LDAP server.
`ldap.compliance`	`[ssl] ssl-compliance`	New, Optional		*none, fips, sp800-131-transition, sp800-131-strict, suite-b-128, suite-b-192*	Sets the compliance level for SSL and TLS connections to the LDAP server. This value is not used when running within a WebSphere JVM because the compliance level is automatically determined based on how WebSphere is configured.
`ldap.ssl-v3-enable`	`[ssl] ssl-v3-enable`	New, Optional	True	*true, false*	Enables or disables the use of SSL version 3 to the LDAP server. For some `ssl.compliance` values, this parameter is always disabled. This parameter is always disabled for compliance levels `sp800-131-strict`, `suite-b-128`, and `suite-b-192`.
`ldap.tls-v10-enable`	`[ssl] tls-v10-enable`	New, Optional	True	*true, false*	Enables or disables the use of TLS version 1.0 to the LDAP server. For some `ssl.compliance` values, this parameter is always disabled. This parameter is always disabled for compliance levels `sp800-131-strict`, `suite-b-128`, and `suite-b-192`.
`ldap.tls-v11-enable`	`[ssl] tls-v11-enable`	New, Optional	True	*true, false*	Enables or disables the use of TLS version 1.1 to the LDAP server. For some `ssl.compliance` values, this parameter is always disabled. This parameter is always disabled for compliance levels `sp800-131-strict`, `suite-b-128`, and `suite-b-192`.

Table 36. Configuration options (continued)

Java Option Name	Existing Comparable Option	Existence in current Java Config	Default	Valid Range	Description
`ldap.tls-v12-enable`	`[ssl] tls-v12-enable`	New, Optional	True	*true, false*	Enables or disables the use of TLS version 1.2 to the LDAP server. For some `ssl.compliance` values, this parameter is always disabled. This parameter is always enabled for `sp800-131-strict`, `suite-b-128`, and `suite-b-192`.
`ldap.cipher-suites`	`[ssl] ssl-v3-cipher-specs, [ssl] tls-v10-cipher-specs,[ssl] tls-v11-cipher-specs,[ssl] tls-v12-cipher-specs`	New, Optional	Java defaults	*[semicolon list of Java cipher names]*	Specifies which cipher suites to use for all SSL and TLS protocols. Example: SSL_DHE_DSS_WITH_3DES_EDE_CBC_SHA; SSL_DHE_DSS_WITH_AES_128_CBC_SHA; SSL_DHE_DSS_WITH_AES_128_CBC_SHA256 For information about the cipher suite names, see http://publib.boulder.ibm.com/infocenter/java7sdk/v7r0/index.jsp?topic=%2Fcom.ibm.java.security.component.doc%2Fsecurity-component%2Fjsse2Docs%2Fciphersuites.html.
`ldap.ssl-truststore`		New, Optional		*Filename string*	The file name of a Java JCEKS keystore that contains the trusted CA signers for the LDAP server Certificate. The API converts the value that is placed in the configuration file into URL format. The API supports only `file:` protocol. If you do not provide `Filename string` in the URL, specify `java.naming.ldap.factory.socket`, if you enabled `ldap.ssl-enable`.
`ldap.ssl-truststore-pwd`		New, Required only if *ldap.ssl-truststore* is specified		*Password string*	The password for the *ldap.ssl-truststore*. This password is obfuscated by `SvrSslCfg` and `RgyConfig` when set. Provide the password if *ldap.ssl-trustsore* is set.
`ldap.login-failures-persistent`	`[ldap] login-failures-persistent`	New, Optional	False	*true, false*	Login failures are used with the three-strikes policy. If you set this option to `false`, each process by using this API stores the number of login failures in memory. If multiple servers are involved, the total number of login failures to trigger a *strike-out* might vary. If you set this option to `true`, the strike count is stored in LDAP and shared across all servers. An accurate count can be kept in a multiserver environment.
`ldap.auth-using-compare`	`[ldap] auth-using-compare`	New, Optional	Defaults vary depending on LDAP server type.	*true, false*	Set this option to false to validate every *dn/password* by using a new connection to LDAP, and a simple bind. Set this option to `true` to compare the LDAP against the password attribute to validate the password. Some LDAP servers do not support this setting and ignores it.
`ldap.bind-dn`	`[ldap] bind-dn`	*New, Required*		*valid LDAP DN string*	The DN to simple bind to LDAP for all management LDAP operations.
`ldap.bind-pwd`	`[ldap] bind-pwd`	*New, Required*		*valid password string*	The LDAP `bind-dn` account password. *SvrSslCfg* and *RgyConfig* obfuscates this value in the configuration file.

Table 36. Configuration options (continued)

Java Option Name	Existing Comparable Option	Existence in current Java Config	Default	Valid Range	Description
`ldap.return-registry-id`	`[ldap] cache-return-registry-id`	*New, Optional*	False	*true, false*	If set to true, `RgyUser.RgyEntity.getId()` returns the Security Access Manager user ID for the specific user that is stored in the LDAP registry. If set to false, `RgyUser.RgyEntity.getId()` returns the Security Access Manager user ID for the user that was passed into the `RgyRegistry.getUser()` method. Security Access Manager IDs are not case-sensitive. The user ID returned differs if the case of the ID passed to `RgyRegistry.getUser()` is different from the case of the value that is stored in LDAP.
`ldap.user-self-care-objectclass`		*New, Optional*	Empty	*valid LDAP objectClass string*	The name of an AUXILLARY objectClass to ensure what user entries have so that self-care attributes can be added to existing and new native user LDAP entries.
`ldap.default-policy-override-support`	`[ldap] default-policy-override-support`	*New, Optional*	False	*true, false*	If set to `true`, the Security Access Manager per-user policy is not used. Instead, the global policy takes effect.
`java.naming.ldap.factory.socket`		*New, Optional*		*name of class*	Makes it possible for the caller to provide their own SSL socket factory to use with JNDI to the LDAP servers.
`ldap.cache-policy-expire-time`	`[ldap] cache-policy-expire-time`	*New, Optional*	600 (seconds)	*0 -> 86400*	The duration in seconds for which the global policy is cached in the memory before being read again from LDAP.
`ldap.max-auth-connections`	`[ldap] max-auth-connections`	*New, Optional*	0	*0 -> 32768*	Non-zero value that sets the number of simultaneous LDAP connections that are used to authenticate users (when *auth-using-compare = false*)
`ldap.group-map-size`			1024	*0 -> Maximum Integer*	The number of entries in a map that is used to convert group native names (DNs) into Security Access Manager IDs. An LRU algorithm to enables creation of new entries.
`ldap.group-map-lifespan`			60	*0 -> 86400*	Duration in seconds for which the entry stays in the map, used to convert group native names (DNs) into Security Access Manager IDs.
`ldap.late-lockout-notification`			False	*true, false*	Notifies the user when the account is locked due to several password login attempts during the $n+1^{th}$ login rather than the n^{th}. Here, `n` is the value of *maxFailedLogins* policy attribute in effect for the user.
`authz.enable-authorization`		*New, Optional*	False	*true, false*	When *LdapRgyRegistryFactory.getRgyRegistryInstance*(URL `propertiesUrl`, `Map enhancements`) is used, it recognizes the option and enables the authorization of the API operations. Provide *authz.pdauthorizatoncontext-user*, used as admin user and authorizes each access.
`authz.pdauthorization context-user`		*New, Optional* (conditional)		*Security Access Manager user ID*	When *authz.enable-authorization* is set, this `user ID` is authorized in API operations. If *authz.pdauthorizatoncontext-pwd* is also specified, then the Security Access Manager user account has an additional purpose. The user account is passed with the password to the construction of the *PDAuthorizationContext* constructed by the API. If required, you can override the joint usage by calling *AuthzRgyRegistryFactory.updateAdminId(RgyRegistry rgyRegistry, String adminUserId)*. Doing so changes the Security Access Manager ID used in the authorization decision.

Table 36. Configuration options (continued)

Java Option Name	Existing Comparable Option	Existence in current Java Config	Default	Valid Range	Description
`authz.pdauthorization context-pwd`		*New, Optional*		*Security Access Manager user password*	If you specify *authz.pdauthorizatoncontext-pwd* along with *authz.pdauthorizatoncontext-user*, the Security Access Manager user and password are passed to the construction of the *PDAuthorizationContext*.This is constructed by the API used to provide authorization decision outcomes for API operations.
`authz.initialize-pdadmin`		*New, Optional*	False	*true,false*	When this option is set to true, the API calls `PDAdmin.initialize()` before creating the `PDAuthorizationContext`. It also calls `PDAdmin.shutdown()` when the API calls `RgyRegistry.close()`.
`authz.enable-audit`		*New, Optional*	False	*true,false*	When you use `LdapRgyRegistryFactory. getRgyRegistryInstance(URL propertiesUrl, Map enhancements)`, it recognizes this option, and enable the API operation auditing. If you do not enable *authz.enable-authorization* option, the user who does this operation is an unauthenticated user.
`authz.audit-file-pattern`		*New, Optional* (conditional)		*File name pattern*	Enables *authz.enable-audit*. Pass this attribute to the Java*java.util.logging.FileHandler* constructor to provide appropriate description for the documentation.
`authz.audit-file-limit`		*New, Optional*	0	*0 -> MAXINTEGER*	Passed to the Java *java.util.logging.FileHandler* constructor so that documentation has the appropriate description.
`authz.audit-file-count`		*New, Optional*	1	*1,8192*	Passed to the Java *java.util.logging.FileHandler* constructor to ensure that the documentation has the appropriate description.
`appsvr-servername`		*Already present, Optional* (conditional)		*string*	Set this option if *authz.enable-audit* is enabled. Use this option to segregate the application by using the new Registry Direct Java API in Java Logger name space for audit logging. For example, The audit names are *com.tivoli.pd.rgy.authz. testapp-tam611.mgmt* and *com.tivoli.pd.rgy.authz.testapp-tam611.authn. testapp-tam611* is the string passed. **Note:** Although the audit logger is listed in the Java Logger name space, it outputs the records into its own file. You can enable or disable the output to the audit log file by increasing or decreasing the Java logging level for the audit logger names.
`authz.authorize-group-list`	`[delegated-admin] authorize-group-list`	*New, Optional*	False	*true, false*	Indicates whether the API must check the authorization on the `listGroup()` and `listNativeGroups()`. This option matches the behavior options of the Policy Server. For example, the *pdadmin group list* and *group list-dn* commands.

Example usage

Creating an instance of RgyRegistry

After creating the configuration file, use this API with the corresponding LDAP account on the existing Security Access Manager registry. If the configuration file is located at `/opt/testapp/testapp.properties`, then the sample configuration file is as follows:

```
URL propertiesUrl = null;
try {
propertiesUrl = new URL("file:///opt/testapp/testapp.properties");
```

```
}
catch (MalformedURLException e) {
e.printStackTrace();
System.exit(1);
}

RgyRegistry rgyRegistry = null;
try {
rgyRegistry = LdapRgyRegistryFactory.getRgyRegistryInstance
(propertiesUrl, null);
}
catch (RgyException e) {
e.printStackTrace();
System.exit(1);
}
```

Ending use of RgyRegistry

Close the `RgyRegistry` instance when your application does not use the instance anymore. The sample usage is as follows:

```
rgyRegistry.close();
```

Groups

Creating a group

The following sample demonstrates how to create a Security Access Manager group called *testgroup* with LDAP DN *cn=testgroup,o=ibm,c=us*.

Example:

```
String groupId = "testgroup";
String groupCn = "testgroup";
String groupNativeId = "cn=testgroup,o=ibm,c=us";
RgyAttributes rgyAttributes = rgyRegistry.newRgyAttributes();
rgyAttributes.putAttribute(RgyAttributes.COMMON_NAME_NAME, groupCn);
rgyAttributes.putAttribute(RgyAttributes.DESCRIPTION_NAME,
"This is a test Group");
RgyGroup rgyGroup = null;
try {
rgyGroup = rgyRegistry.createGroup("Default", groupId, groupNativeId,
rgyAttributes);
}
catch (RgyException e) {
e.printStackTrace();
System.exit(1);
}
```

This example obtains a `RgyAttributes` instance to create the group and set the group attributes, assuming that the Security Access Manager domain is default and that it contains a suffix `o=ibm,c=us` in LDAP.

Showing group details

The following sample displays the information about the group. This function is equivalent of pdadmin *group show testgroup*:

```
// Fetch the group
String groupId = "testgroup";
RgyGroup rgyGroup = null;
try {
```

```
 rgyGroup = rgyRegistry.getGroup("Default", groupId);
}
catch (RgyException e) {
 e.printStackTrace();
 System.exit(1);
}
// Ensure the group was found
if (rgyGroup == null) {
 System.out.println("Group does not exist");
 System.exit(1);
}
System.out.println("Group ID: "+rgyGroup.getId());
System.out.println("LDAP DN: "+rgyGroup.getNativeId());
String description =
String) rgyGroup. getOneAttributeValue(RgyAttributes.DESCRIPTION_NAME);
if (description == null) {
 description = "";
}
System.out.println("Description: "+description);
System.out.println
("LDAP CN: "+rgyGroup. getOneAttributeValue(RgyAttributes.COMMON_NAME_NAME);
String isSecEntity =
String) rgyGroup.getOneAttributeValue(RgyAttributes. IS_SEC_ENTITY_NAME);
if (isSecEntity.equalsCaseIgnore(RgyAttributes. BOOL_TRUE_VALUE)) {
 isSecEntity = "Yes";
}
else {
 isSecEntity = "No";
}
System.out.println("Is SecGroup: "+isSecEntity);
```

Deleting a group

The following sample code displays the steps to delete the Security Access Manager group `testgroup` from the Security Access Manager domain `Default` and remove the native LDAP entry.

The last parameter of `deleteGroup()` is set to true.

If the native LDAP entry is also a member of another Security Access Manager domain, or another application places child entries, the removal of the native LDAP entry fails. Despite this error, the Security Access Manager component is still removed.

```
String groupId = "testgroup";
try {
 rgyRegistry.deleteGroup("Default", groupId, true);
}
catch (WarningNativeEntityInUseRgyException e) {
 System.out.println("Warning: unable to remove native LDAP entry");
 // Ignore
}
catch (RgyException e) {
 e.printStackTrace();
 System.exit(1);
}
```

Importing a native group

Existing native LDAP groups can be imported as Security Access Manager groups. Obtain the native LDAP group information as a`RgyGroup` instance, then invoke the `RgyGroup.importNativeGroup()` method.

The following sample shows the sample code to import the native group as Security Access Manager group:

```
// First fetch the Native Entity
String groupNativeId = "cn=testgroup,o=ibm,c=us";
RgyGroup rgyGroup = null;
try {
rgyGroup = rgyRegistry.getNativeGroup("Default", groupNativeId);
}
catch (RgyException e) {
 e.printStackTrace();
 System.exit(1);
}
// Ensure the group was found
if (rgyGroup == null) {
 System.out.println("Group does not exist");
 System.exit(1);
}
// Only import, if not already a TAM group (isSecEntity=FALSE)
String isSecEntity = (String) rgyGroup.getOneAttributeValue
(RgyAttributes. IS_SEC_ENTITY_NAME);
if (isSecEntity.equalsIgnoreCase(RgyAttributes. BOOL_FALSE_VALUE)) {
 String groupId = "testgroup";
 RgyAttributes rgyAttributes = rgyRegistry.newRgyAttributes();
 try {
 rgyGroup.importNativeGroup(groupId, rgyAttributes);
 }
 catch (RgyException e) {
  e.printStackTrace();
  System.exit(1);
 }
}
```

Listing group members

The following sample displays the sample code to list the members belonging to a specific group. Ensure that you fetch the group before you list its members. The members are returned as a set of string values.

```
// Fetch the group
String groupId = "testgroup";
RgyGroup rgyGroup = null;
try {
 rgyGroup = rgyRegistry.getGroup("Default", groupId);
}
catch (RgyException e) {
 e.printStackTrace();
 System.exit(1);
}
// Ensure the group was found
if (rgyGroup == null) {
 System.out.println("Group does not exist");
 System.exit(1);
}
// Get Set of group's member IDs
Set userIds = null;
try {
 userIds = rgyGroup.listMemberIds();
}
catch (RgyException e) {
 e.printStackTrace();
 System.exit(1);
}
// Display the member IDs
System.out.print("Members=");
Iterator iter = userIds.iterator();
```

```
while (iter.hasNext()) {
 String userId = (String) iter.next();
 System.out.print(userId+", ");
}
System.out.println();
```

A list of the groups members as native user IDs can be obtained using `rgyGroup.listMemberNativeIds()`. The native ID Set includes non-TAM users.

Add or remove group members

The following example displays the code to add or remove the group members. Before modifying, ensure that you fetch the user.

```
// Fetch the group
String groupId = "testgroup";
RgyGroup rgyGroup = null;
try {
 rgyGroup = rgyRegistry.getGroup("Default", groupId);
}
catch (RgyException e) {
 e.printStackTrace();
 System.exit(1);
}
// Ensure the group was found
if (rgyGroup == null) {
 System.out.println("Group does not exist");
 System.exit(1);
}
// Create a List of user IDs and add to group
List userIds = new ArrayList();
userIds.add("testuser");
try {
 rgyGroup.addMembers(userIds);
}
catch (RgyException e) {
 e.printStackTrace();
 System.exit(1);
}
// Remove List of user IDs from the group
try {
 rgyGroup.removeMembers(userIds);
}
catch (RgyException e) {
 e.printStackTrace();
 System.exit(1);
}
```

Modifying group attribute

The following example demonstrates how to modify the groups description attribute.

Before you modify a group, you must fetch the group.

Groups do not have many attributes that can be modified.

```
// Fetch the group
String groupId = "testgroup";
RgyGroup rgyGroup = null;
try {
 rgyGroup = rgyRegistry.getGroup("Default", groupId);
}
catch (RgyException e) {
 e.printStackTrace();
```

```
 System.exit(1);
}
// Ensure the group was found
if (rgyGroup == null) {
 System.out.println("Group does not exist");
 System.exit(1);
}
// Replace the current attribute
try {
 rgyGroup.attributeReplace(RgyAttributes.DESCRIPTION_NAME,
"Replacement Description");
}
catch (RgyException e) {
 e.printStackTrace();
 System.exit(1);
}
// Remove the description attribute
try {
 rgyGroup.attributeDelete(RgyAttributes.DESCRIPTION_NAME);
}
catch (RgyException e) {
 e.printStackTrace();
 System.exit(1);
}
```

Users and per-user policy

Assuming that the Security Access Manager domain is `Default` and that there is a suffix `o=ibm,c=us` in LDAP, the following code creates a Security Access Manager user called `testuser`. This `testuser` has the LDAP DN `cn=testuser,o=ibm,c=us`. First, obtain an `RgyAttributes` instance to set the users attributes and then create the user. A few optional attributes set for the user during the time of creation include account and password policy and account state.

```
// Setup the required attributes
RgyAttributes rgyAttributes = registry.newRgyAttributes();
rgyAttributes.putAttribute(RgyAttributes.COMMON_NAME_NAME,"testuser");
rgyAttributes.putAttribute(RgyAttributes.SURNAME_NAME, "user");
// Setup some optional attributes
rgyAttributes.putAttribute(RgyAttributes.MIN_PASSWORD_LENGTH_NAME, "8");
rgyAttributes.putAttribute(RgyAttributes.MAX_LOGIN_FAILURES_NAME, "2");
rgyAttributes.putAttribute(RgyAttributes.DISABLE_TIME_INTERVAL_NAME, "0");
Date currentTime = new Date();
currentTime.setTime(currentTime.getTime() + (3600 * 1000L));
String registryDate = null;
try {
 registryDate = registry.toRegistryDate(currentTime);
}
catch (RgyException e) {
 e.printStackTrace();
 System.exit(1);
}
rgyAttributes.putAttribute(RgyAttributes.ACCOUNT_EXPIRY_DATE_NAME,
registryDate);
rgyAttributes.putAttribute(RgyAttributes.SEC_ACCT_VALID_NAME,
RgyAttributes.BOOL_TRUE_VALUE);
// Create the user
String userId = "testuser";
String userNativeId = "cn=testuser,o=ibm,c=us";
RgyUser rgyUser = null;
try {
 user = registry.createUser("Default", userId, userNativeId, "passw0rd".
toCharArray(), true, rgyAttributes, null);
}
catch (RgyException e) {
```

```
 e.printStackTrace();
 System.out.println("FAILED: Unable to create user: "+id);
 System.exit(1);
}
```

Showing user details

This is the equivalent of pdadmin `user show testuser` command. The following example displays the user details such as User ID, Native ID, Common name, Surname, and descriptive name.

```
RgyUser rgyUser = null;
try {
 rgyUser = registry.getUser(null, userId);
}
catch (RgyException e) {
 e.printStackTrace();
 System.exit(1);
}
if (rgyUser == null) {
 System.out.println("User not found");
 System.exit(1);
}
System.out.println("Login ID: "+rgyUser.getId());
System.out.println("LDAP DN: "+rgyUser.getNativeId());
System.out.println("LDAP CN: "+rgyUer.getOneAttributeValue
(RgyAttributes.COMMON_NAME_NAME);
System.out.println("LDAP SN: "+rgyUser.getOneAttributeValue
(RgyAttributes.SURNAME_NAME);
String description = (String) rgyUser.getOneAttributeValue
(RgyAttributes.DESCRIPTION_NAME);
if (description == null) {
 description = "";
}
System.out.println("Description: "+description);
System.out.println("Is SecUser: "+yesNo
(rgyUser, RgyAttributes. IS_SEC_ENTITY_NAME);
System.out.println("Is GSO user: "+yesNo
(rgyUser, RgyAttributes. IS_GSO_USER_NAME));
System.out.println("Account valid: "+yesNo
(rgyUser, RgyAttributes. SEC_ACCT_VALID_NAME));
System.out.println("Password valid: "+yesNo
(rgyUser, RgyAttributes. SEC_PWD_VALID_NAME));

//-----

String yesNo(RgyEntity rgyEntity, String attributeName)
{
 String value = (String) rgyGroup.getOneAttributeValue(attributeName);
 if (value.equalsCaseIgnore(RgyAttributes. BOOL_TRUE_VALUE)) {
  value = "Yes";
 }
 else {
  value = "No";
 }
 return value;
}
```

Deleting a user

If the native LDAP entry is a member of another Security Access Manager domain, or another application has placed child entries beneath it, the removal of the native LDAP entry fails. An exception is thrown from the API but the Security Access Manager component is still removed.

The following example deletes the Security Access Manager user `testuser` from the `Default` Security Access Manager domain and removes the native LDAP entry. Last parameter of `deleteUser()` is set to true.

```
String userId = "testuser";
try {
 rgyRegistry.deleteUser("Default", userId, true);
}
catch (WarningNativeEntityInUseRgyException e) {
 System.out.println("Warning: unable to remove native LDAP entry");
 // Ignore
}
catch (RgyException e) {
 e.printStackTrace();
 System.exit(1);
}
```

Importing a native user

You can import the existing native LDAP users as Security Access Manager users. The native LDAP user information must be obtained as an `RgyUser` instance, then the `RgyUser.importNativeGroup()` method can be invoked. The following example imports a `Default` native user as a Security Access Manager user `testuser` from the native LDAP entry. Last parameter of `deleteUser()` is set to true.

```
/ First fetch the Native Entity
String userNativeId = "cn=testuser,o=ibm,c=us";
RgyUser rgyUser = null;
try {
rgyUser = rgyRegistry.getNativeUser("Default", userNativeId);
}
catch (RgyException e) {
 e.printStackTrace();
 System.exit(1);
}
// Ensure the Native user was found
if (rgyUser == null) {
 System.out.println("Group does not exist");
 System.exit(1);
}
// Only import, if not already a TAM user (isSecEntity=FALSE)
String isSecEntity = (String) rgyUser.getOneAttributeValue
(RgyAttributes. IS_SEC_ENTITY_NAME);
if (isSecEntity.equalsIgnoreCase(RgyAttributes. BOOL_FALSE_VALUE)) {
 String userId = "testuser";
 RgyAttributes rgyAttributes = rgyRegistry.newRgyAttributes();
 // Setup an optional attribute
 rgyAttributes.putAttribute(RgyAttributes.SEC_ACCT_VALID_NAME,
  RgyAttributes.BOOL_TRUE_VALUE);
 try {
  rgyUser.importNativeUser(userId, rgyAttributes, null);
 }
 catch (RgyException e) {
  e.printStackTrace();
  System.exit(1);
 }
}
```

Listing a user's group memberships

Fetch the user before listing its group memberships. Returns the groups the user belongs to as a set of string:

```
// Fetch the user
String userId = "testuser";
RgyUser rgyUser = null;
try {
 rgyUser = rgyRegistry.getUser("Default", userId);
}
catch (RgyException e) {
 e.printStackTrace();
 System.exit(1);
}
// Ensure the user was found
if (rgyUser == null) {
 System.out.println("Group does not exist");
 System.exit(1);
}
// Get Set of group IDs the user is a memberof
Set groupIds = null;
try {
 groupIds = rgyUser.listGroups();
}
catch (RgyException e) {
 e.printStackTrace();
 System.exit(1);
}
// Display the group IDs
System.out.print("Groups=");
Iterator iter = GroupIds.iterator();
while (iter.hasNext()) {
 String groupId = (String) iter.next();
 System.out.print(groupId+", ");
}
System.out.println();
```

A list of the user group memberships as group native ID can also be obtained using `rgyUser.listNativeGroups ()` instead. The native ID set can include groups that are not Security Access Manager groups.

Modifying user attributes

The following example shows the equivalent of the pdadmin commands `user modify testuser password-valid yes`, `policy set max-login-failures unset -user testuser`, and `policy set min-password-length 7 -user testuser`.

Ensure that you fetch the user before you modify the user attributes.

```
// Fetch the user
String userId = "testuser";
RgyUser rgyUser = null;
try {
 rgyUser = rgyRegistry.getUser("Default", userId);
}
catch (RgyException e) {
 e.printStackTrace();
 System.exit(1);
}
// Ensure the user was found
if (rgyUser == null) {
 System.out.println("Group does not exist");
 System.exit(1);
}
// Set the password-valid
try {
 rgyUser.attributeReplace(RgyAttributes.SEC_PWD_VALID_NAME,
RgyAttributes.BOOL_TRUE_VALUE);
}
catch (RgyException e) {
```

```
 e.printStackTrace();
 System.exit(1);
}
// Unset the max-login-failures policy
try {
 rgyUser.attributeDelete(RgyAttributes. MAX_LOGIN_FAILURES_NAME);
}
catch (RgyException e) {
 e.printStackTrace();
 System.exit(1);
}
// Set the min-password-length policy to 7
try {
 rgyUser.attributeReplace(RgyAttributes.MIN_PASSWORD_LENGTH_NAME, "7");
}
catch (RgyException e) {
 e.printStackTrace();
 System.exit(1);
}
```

Resetting the user password

This action is the equivalent to `pdamin` command `user modify testuser password changeme`.

The following example shows how to perform an administrative reset of the user password:

```
// Fetch the user
String userId = testuser;
RgyUser rgyUser = null;
try {
 rgyUser = rgyRegistry.getUser(Default, userId);
}
catch (RgyException e) {
 e.printStackTrace();
 System.exit(1);
}
// Ensure the user was found
if (rgyUser == null) {
 System.out.println(Group does not exist);
 System.exit(1);
}
try {
 rgyUser = rgyUser.setPassword(changeme.toCharArray());
}
catch (RgyException e) {
 e.printStackTrace();
 System.exit(1);
}
```

Changing the user password

To change the user password, authenticate the old password and request to set a new password.

This operation is similar to a `pkmspasswd` of WebSEAL. The following example shows the sample code for changing the `oldpassword` to `newpassword` for a user `testuser`:

```
// Fetch the user
String userId = "testuser";
RgyUser rgyUser = null;
try {
 rgyUser = rgyRegistry.getUser("Default", userId);
}
```

```
catch (RgyException e) {
 e.printStackTrace();
 System.exit(1);
}
// Ensure the user was found
if (rgyUser == null) {
 System.out.println("Group does not exist");
 System.exit(1);
}
try {
 rgyUser = rgyUser.changePassword("oldpassword".toCharArray(),
 "newpassword".toCharArray());
}
catch (RgyException e) {
 e.printStackTrace();
 System.exit(1);
}
```

Authenticating the user Password

This section provides an example showing the authentication script.

The following example shows about authentication the `default` user password `passw0rd` .

```
// Fetch the user
String userId = "testuser";
RgyUser rgyUser = null;
try {
 rgyUser = rgyRegistry.getUser("Default", userId);
}
catch (RgyException e) {
 e.printStackTrace();
 System.exit(1);
}
// Ensure the user was found
if (rgyUser == null) {
 System.out.println("Group does not exist");
 System.exit(1);
}
try {
 rgyUser = rgyUser.authenticate("passw0rd".toCharArray());
}
catch (RgyException e) {
 e.printStackTrace();
 System.exit(1);
}
```

Appendix E. User registry differences

Each user registry presents unique concerns when integrated with Security Access Manager. This release of Security Access Manager supports LDAP and URAF user registries.

Security Access Manager supports the following LDAP user registries:

- Tivoli Directory Server
- IBM z/OS Security Server LDAP Server
- Microsoft Active Directory Lightweight Directory Services (AD LDS) (Windows 2008)
- Novell eDirectory Server
- Sun Java System Directory Server, version 7.0

Security Access Manager supports the following URAF user registries:

- Microsoft Active Directory Server

General concerns

The following concerns are specific to all the supported user registries:

- Avoid using the forward slash (/) character when defining the names for users and groups when that name is defined using distinguished names strings. Each user registry treats this character differently.
- Avoid using leading and trailing blanks in user and group names. Each user registry treats blanks in a different manner.

LDAP concerns

The following concerns are specific to all the supported LDAP user registries:

- There are no configuration steps required for Security Access Manager to support the Password Policy of LDAP. Security Access Manager does not assume the existence or non-existence of the Password Policy of the LDAP at all.

  Security Access Manager enforces its own Password Policy first. Security Access Manager attempts to update password in LDAP only when the provided password passes Password Policy check of Security Access Manager.

  After that, Security Access Manager tries to accommodate the Password Policy of LDAP to the best of its ability using the return code that it gets from LDAP during a password-related update.

  If Security Access Manager can map the return code without any ambiguity with the corresponding Security Access Manager error code, it does so and returns a proper error message.
- To take advantage of the multi-domain support in Security Access Manager, you must use an LDAP user registry. When using a URAF user registry, only a single Security Access Manager domain is supported.
- When using an LDAP user registry, the capability to own global sign-on credentials must be explicitly granted to a user. After this capability is granted, it can be removed. Conversely, users that are created in a URAF user registry are automatically given this capability. This capability cannot be removed.

- Leading and trailing blanks in user names and group names are ignored when using an LDAP user registry in a Security Access Manager secure domain. To ensure consistent processing regardless of the user registry, define user names and group names without leading or trailing blanks.
- Attempting to add a single duplicate user to a group does not produce an error when using an LDAP user registry.
- The Security Access Manager authorization API provides a credential attribute entitlements service. This service retrieves user attributes from a user registry. When this service is used with an LDAP user registry, the retrieved attributes can be string data or binary data. However, when used with a URAF user registry, the retrieved attributes can be string data, binary data, or integer data.

Sun Java System Directory Server concerns

The following task describes how to modify the default value for the look-through limit on the directory server.

About this task

The following concerns are specific to Sun Java System Directory Server:

If the user registry contains more entries than the defined look-through limit, the directory server might return the following status that Security Access Manager treats as an error:

```
LDAP_ADMINLIMIT_EXCEEDED
```

When the directory server is installed, the default value is 5000. To modify this value, perform the following steps from the Sun Java System Directory Server Console:

Procedure

1. Select the **Configuration** tab.
2. Expand the **Data** entry.
3. Select **Database Settings**.
4. Select the **LDBM Plug-in Settings** tab.
5. In the **Look-through Limit** field, type the maximum number of entries that you want the server to check in response to the search, or type `-1` to define no maximum limit.

   If you bind the directory as the Directory Manager, the look-through limit is unlimited and overrides any settings specified in this field.

Microsoft Active Directory Lightweight Directory Service (AD LDS) concerns

This section describes concerns specific to Microsoft Active Directory Lightweight Directory Service (AD LDS).

The following concerns are specific to AD LDS.

- Policy Server configuration allows you to select between a standard or minimal data model for the user registry. Because AD LDS allows only a single naming attribute to be used when creating LDAP objects, AD LDS requires the minimal data model. Regardless of which data model is chosen during Policy Server configuration, Security Access Manager will always use the minimal data model when AD LDS is selected as the user registry.

- Because the common name (cn) value in AD LDS must be single valued, the value specified for the `cn` attribute must be the same value used for the distinguished name (dn) when a user or group is created and `cn` is used as the naming attribute in the dn; for example, the following command to create a user would NOT be allowed:

  ```
  pdadmin user create user1 cn=user1,o=ibm,c=us fred user1 password1
  ```

  In the example, the `cn` value, `fred`, is different from the `cn` naming attribute in the `dn`, `user1`.

URAF concerns

The following concerns are specific to all the supported URAF user registries:

- When you use a URAF user registry, only a single Security Access Manager domain is supported. To take advantage of the Security Access Manager multi-domain support, use an LDAP user registry.
- Users that are created in a URAF user registry are automatically given the capability to own global sign-on credentials. This capability cannot be removed. When you use an LDAP user registry, this capability must be explicitly granted. After this capability is granted, it can later be removed.
- The Security Access Manager authorization API provides credentials attribute entitlements service. This service is used to retrieve user attributes from a user registry. When this service is used with a URAF user registry, the retrieved attributes can be string data, binary data, or integer data. However, when used with an LDAP user registry, the retrieved attributes can be only string data or binary data.

Microsoft Active Directory Server concerns

In addition to the general URAF concerns, the following concerns are specific to Microsoft Active Directory Server:

- Users that are created in Active Directory might have an associated primary group. The Active Directory default primary group is Domain Users.

  However, the Active Directory does not add the primary group information to the user `memberOf` or the group `member` attribute. When Security Access Manager queries for a list of group members, the result does not include any members who belong to the primary group. When Security Access Manager queries for all the groups where the user belongs, the result does not display the primary group of the user.

  For this reason, do not use a Security Access Manager group as the Active Directory primary group for Security Access Manager users.
- Security Access Manager does not support cross domain group membership or universal groups. Security Access Manager does not support importing these types of groups.
- When Security Access Manager imports a dynamic group, the `ivacld-servers` and `remote-acl-users` groups apply read permission on each authorization store to which the dynamic group belongs.

  The read permission enables Security Access Manager blade servers, such as WebSEAL, to read permission to the registry authorization store. Thus, the blade server reads dynamic group data, such as group membership for building Security Access Manager credentials.

  Manually removing the read permission while Security Access Manager is configured to the Active Directory registry results in adverse behavior, such as inaccurate group membership.

- If the option to change the user password by using LDAP APIs is enabled in an environment where:
  - Security Access Manager is configured to use the Active Directory user registry, and
  - Security Access Manager blade servers use LDAP APIs to communicate with the Active Directory server

  then Security Access Manager must be configured with Secure Socket Layer (SSL) to allow connections between the LDAP client and the Active Directory server. The Active Directory environment must also be enabled to accept LDAP connections over Secure Socket Layer (SSL).
- When you use an Active Directory user registry in a Security Access Manager configuration with blade servers that use LDAP APIs to communicate with the Active Directory server, Security Access Manager supports user password change requests by using either the Policy Server or LDAP APIs. Change user password requests by using the LDAP APIs do not require the Policy Server to run.

  The use of LDAP APIs to communicate with the Active Directory Server for blade servers is a multiplatform support that allows blade servers to be installed on systems that are not clients of the same domain as the policy server. In this configuration, the policy server must be installed and configured on a Windows operating system.
- When you use an Active Directory user registry, each user name and each group name in a domain must be unique. User and group short name values that are stored in the `sAMAccountName` attribute of Active Directory user objects and group objects. Active Directory user objects and group objects both have the `sAMAccountName` attribute as one of their attributes. Microsoft requires that the `sAMAccountName` attributes be unique within an Active Directory domain.
- When you use a multi-domain Active Directory user registry, multiple users and groups can be defined with the same short name if they are in different domains. However, the full name of the user or group, including the domain suffix, must always be specified to Security Access Manager.
- The following items are ignored when you use Microsoft Active Directory Server as the user registry in a Security Access Manager secure domain:
  - Leading and trailing blanks in user names
  - Group names

  To ensure consistent processing regardless of the user registry, define user names and group names without leading or trailing blanks.
- Security Access Manager supports the use of an email address or other formats of the `userPrincipalName` attribute of the Active Directory registry user object as a Security Access Manager user identity. When the option is enabled, both the default and the email address or another `userPrincipalName` format can co-exist in the Security Access Manager environment.

  The default format of the `userPrincipalName` registry attribute is `user_id@`*`domain_suffix`*, where *domain_suffix* is the Active Directory domain where the user identity is created.

  For example, `johndoe@example.com` is the value of the `userPrincipalName`; `example.com` is the Active Directory domain where the user identity is created. The Security Access Manager user identity corresponding to the registry user in this example is either`johndoe@example.com` or `johndoe`. It depends on whether Security Access Manager is configured to use Active Directory with multiple domains or a single domain.

The alternative format of the `userPrincipalName` attribute is `user_id@any_suffix`. `any_suffix` can be any domain (Active Directory or non-Active Directory) other than the Active Directory domain in which the user identity is created.

For example, if the registry user `johndoe@other_domain.com` is created in Active Directory `example.com`, and the registry user `johndoe@example.com` is created in Active Directory domain `child_domain.example.com`. Both users can be Security Access Manager users, and their user identities are `johndoe@other_domain.com` and `johndoe@example.com`.

Enable the alternative user principal name (UPN) support in all Security Access Manager runtime environments. Doing so ensures that Security Access Manager user identities work properly with alternative UPNs.

When the use of alternative UPN format as user identity is enabled, it cannot be reversed without breaking Security Access Manager functions.

- Although users and groups can be created with names that use a distinguished name string, subsequent operations on the object might fail. A distinguished name string contains a forward slash (/) character. Some Active Directory functions interpret the forward slash character as a separator between the object name and the host name. To avoid the problem, do not use a forward slash character to define the user.

Length of names

The maximum lengths of various names that are associated with Security Access Manager vary depending on the user registry that is being used.

See the Table 37 section for a comparison of the maximum lengths that are allowed and the maximum length to use to ensure compatibility with all the user registries that are supported by Security Access Manager.

Table 37. Maximum lengths for names by user registry and the optimal length across user registries

Name	IBM Tivoli Directory Server	IBM z/OS Security Server	Novell eDirectory Server	Sun Java System Directory Server	Microsoft Active Directory Server	Active Directory Lightweight Directory Server (AD LDS)	Optimal length
First name (LDAP CN)	256	256	64	256	64	64	64
Middle name	128	128	128	128	64	64	64
Last name (surname)	128	128	128	128	64	64	64
Registry uid (LDAP DN)	1024	1024	1024	1024	2048	1024	255
Security Access Manager user identity	256	256	256	256	64	64	64
User password	unlimited	unlimited	unlimited	unlimited	256	128	256
User description	1024					1024	1024
Group name	256	256	256	256	64	64	64

Table 37. Maximum lengths for names by user registry and the optimal length across user registries (continued)

Name	IBM Tivoli Directory Server	IBM z/OS Security Server	Novell eDirectory Server	Sun Java System Directory Server	Microsoft Active Directory Server	Active Directory Lightweight Directory Server (AD LDS)	Optimal length
Group description	1024					1024	1024
Single sign-on resource name	240	240	240	240	60	240	60
Single sign-on resource description	1024					1024	1024
Single sign-on user ID	240	240	240	240	60	240	60
Single sign-on password	unlimited	unlimited	unlimited	unlimited	256	unlimited	256
Single sign-on group name	240	240	240	240	60	240	60
Single sign-on group description	1024					1024	1024
Action name	1					1	1
Action description, action type	unlimited					unlimited	unlimited
Object name, object description	unlimited					unlimited	unlimited
Object space name, object space description	unlimited					unlimited	unlimited
ACL name, ACL descriptions	unlimited					unlimited	unlimited
POP name, POP description	unlimited					unlimited	unlimited

Although, the maximum length of an Active Directory distinguished name (registry uid) is 2048, the maximum length of each relative distinguished name (RDN) is 64.

If you configure Security Access Manager to use multiple Active Directory domains, the maximum length of the user identity and group name does not include the domain suffix.

When you use multiple domains, the format of a user identity is *user_id@domain_suffix*.

The maximum length of 64 characters applies only to the `user_id` portion. If you use an email address or other format for the Security Access Manager user identity in the Active Directory, the maximum name length remains the same, but includes the suffix.

Although the lengths of some names can be unlimited, excessive lengths might result in a policy that is difficult to manage. A policy that is difficult to manage might result in poor system performance. Choose maximum values that are logical for your environment.

Notices

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

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

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

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

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan, Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law :

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement might not apply to you.

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

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

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

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

IBM Corporation
2Z4A/101
11400 Burnet Road
Austin, TX 78758 U.S.A.

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

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

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

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

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

All IBM prices shown are IBM's suggested retail prices, are current and are subject to change without notice. Dealer prices may vary.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

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

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to

IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

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

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

If you are viewing this information in softcopy form, the photographs and color illustrations might not be displayed.

Trademarks

IBM, the IBM logo, and ibm.com® are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at www.ibm.com/legal/copytrade.shtml.

Adobe, Acrobat, PostScript and all Adobe-based trademarks are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, other countries, or both.

IT Infrastructure Library is a registered trademark of the Central Computer and Telecommunications Agency which is now part of the Office of Government Commerce.

Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Linux is a trademark of Linus Torvalds 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.

ITIL is a registered trademark, and a registered community trademark of the Office of Government Commerce, and is registered in the U.S. Patent and Trademark Office.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Cell Broadband Engine and Cell/B.E. are trademarks of Sony Computer Entertainment, Inc., in the United States, other countries, or both and is used under license therefrom.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Index

K

L

M

N

O

P

R

S

T

U

W

Printed in USA

SC23-6514-02