CA SiteMinder®

Developer's Guide for Java

r6.0 SP6/r6.x QMR6

Second Edition

CA Product References

This document references the following CA Technologies products:

CA SiteMinder®

Contact CA Technologies

Contact CA Support

For your convenience, CA Technologies provides one site where you can access the information you need for your Home Office, Small Business, and Enterprise CA Technologies products. At http://ca.com/support, you can access the following:

- Online and telephone contact information for technical assistance and customer services
- Information about user communities and forums
- Product and documentation downloads
- CA Support policies and guidelines
- Other helpful resources appropriate for your product

Provide Feedback

If you have comments or questions about CA Technologies product documentation, you can send a message to techpubs@ca.com.

If you would like to provide feedback about CA Technologies product documentation, complete our short customer survey, which is available on the CA Support website at http://ca.com/docs.

Contents

Chapter 1: Java API Overview

This section contains the following topics:

Purpose of the Java APIs

The SiteMinder SDK provides Java APIs for performing the following tasks:

- Creating SiteMinder Agents
- Creating Policy Management applications
- Creating Delegated Management Services (DMS) applications

Installation Path

The Java APIs, documentation, and samples are installed to the following location:

- UNIX platforms: <install_path>/sdk
- Windows platforms: <install_path>\sdk

<install_path> refers to the installation path where you installed the SDK software.

Code Samples

The SiteMinder SDK includes tested samples of SiteMinder client applications. The source files for these samples are located as follows:

- UNIX platforms:

  <install_path>/sdk/samples/<api-name>

- Windows platforms:

  <install_path>\sdk\samples\<api-name>

Notes on the Java samples:

- The samples use properties defined in smjsdksample.properties, located in /sdk/properties. Before you run the Java samples, modify this file with settings for your environment.
- The smjsdksample.properties file also externalizes literal strings used for logging.
- The samples smjavaagentapi and javadmsapi use the policy store created by the sample javapolicyapi. Run smjavapolicyapi before running smjavaagentapi or javadmsapi.
- All the samples use the same logging options and output format.
- When executing the Java samples on a 64-bit UNIX operating environment, note the following:
  - Use a 64-bit JVM. Running the 64-bit samples using a 32-bit JVM is not supported.
  - If you are using the file java-run.sh (in the samples/smjavaagentapi folder) remove the comment indicator before the -d64 flag. Leave this flag commented out when on a 32-bit operating environment.

Policy Server Prerequisite

You must have the SiteMinder Policy Server to run the applications and Policy Server plug-ins that you develop with the SiteMinder SDK.However, in most cases, you do not use the Policy Server to build those applications and plug-ins. The application runtime files can either be local or remote to the Policy Server.

Java Components of the SiteMinder SDK

The Java components of the SiteMinder SDK are listed in the following table:

API Name: **Package Name**	**Primary Interfaces and Classes**
Java Agent API: netegrity.siteminder.javaagent	AgentAPI
Policy Management API: com.netegrity.sdk.policyapi	SmPolicyApi, implemented by SmPolicyApiImpl
Authentication and Authorization APIs com.netegrity.policyserver.smapi	SmAuthScheme (Authentication API) ActiveExpression (Authorization API)
Delegated Management Services API: com.netegrity.sdk.dmsapi	SmDmsApi, implemented by SmDmsApiImpl
Utilities package: com.netegrity.sdk.apiutil	SmApiConnection SmApiSession

Java Agent API

Use the Java Agent API to build a custom agent for enforcing access control and for managing user sessions. Enforcing access control consists of the following:

- User authentication
- User authorization
- Auditing

When using the Java Agent API, code your agent in Java. The Agent API performs the connection to the Policy Server.

The SiteMinder Java Agent API has two implementations:

- The JNI Java Agent API, which relies on the native C/C++ Agent API libraries. This implementation uses the interface presented in the SiteMinder SDK, versions 5.x and later.
- The pure Java Agent API, which replaces the native code used in the JNI Java Agent API with pure Java components. The present version of this API uses the same interface as the JNI Java Agent API.

Because there are no native-mode components, the pure Java Agent API is highly portable to new operating environments. Applications written using the pure Java implementation require certification only against the Java Virtual Machine hosting the implementation, rather than against individual operating systems.

I**mportant!** To use the pure Java API, your Java JCE must be unlimited strength jurisdiction. You can download the files from Sun at the following URL:

http://java.sun.com/products/jce/javase.html#UnlimitedDownload.

Policy Management API

Use the Policy Management API to manage the following SiteMinder elements:

- Resources
- Policies
- Caches
- Security roles

You can manage policies by creating, deleting, associating, and modifying policy objects such as policy domains, realms, and policies.

Authentication API

Use the Authentication API to create a custom authentication scheme that implements authentication service not offered by any of the standard SiteMinder authentication schemes.

Authorization API

Use the Authorization API to implement custom functionality for controlling access to protected resources. The functionality is provided through custom Java classes that are referenced in Policy Server active expressions. An active expression is a string of variable definitions that appears in the following Policy Server objects:

- Active policy
- Active response
- Active rule

Delegated Management Services API

Use the Delegated Management Services (DMS) API to perform tasks such as:

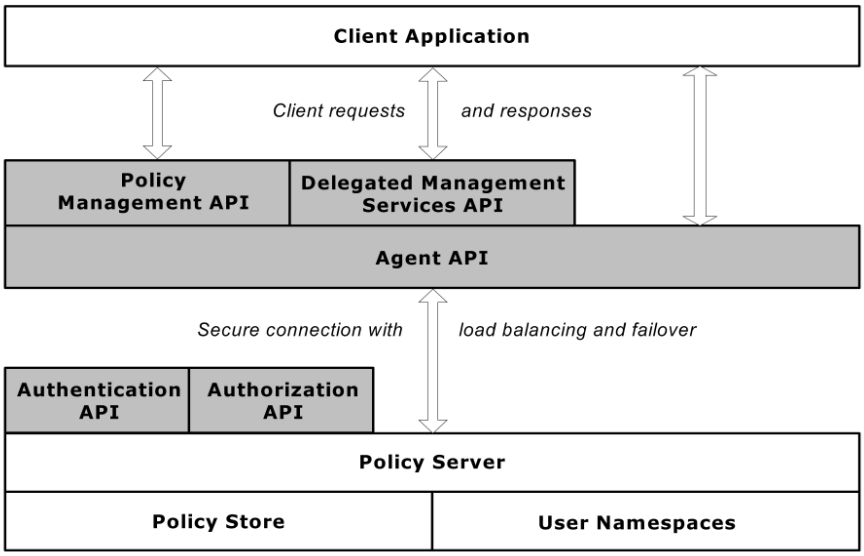
- Manage directory entries
- Grant privileges to users and to groups
- Grant DMS roles to users and to groups

Utilities Package

Use the methods in the Utilities package to implement the APIs in the Policy Management API and the DMS API.

How Java Components Fit Together

The following figure shows how the Java components of the SiteMinder SDK fit together:

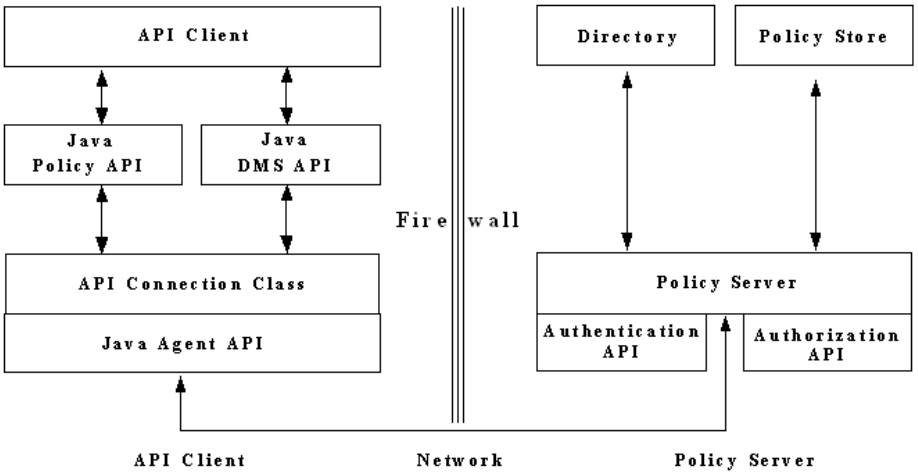


Network Architecture

You can use the Java APIs to write client applications that connect to a remote SiteMinder Policy Server. These applications have access to the following built-in functionality of the Java Agent API:

- Security
- Load balancing
- Failover

The network architecture of the Java APIs is shown in the following figure:



The Policy Management API and the DMS API use the Java Agent API to access the Policy Server. A single API client instance makes a single, secure, Agent API connection to the SiteMinder Policy Server. As long as they share the same process space, multiple API clients can use a single Agent API connection. For example, you can use the Java Agent API to establish a connection, then use that connection to make DMS API calls.

Java API Flow

The steps listed following are required when you are creating a client application with the Policy Management API or the DMS API:

1. Establish a Connection to the Policy Server.
2. Obtain a Session.
3. Make API Requests.
4. Handle Results and Exceptions.

Establish a Connection to the Policy Server

To establish a connection to the Policy Server, use the SmApiConnection class of the Utilities package. This class holds the Agent API handle through which Java API requests are sent.

There are two types of connection handles in this class:

- A *default* connection handle. A default connection handle:
  - Represents a single instance of an Agent API object.
  - Is static across the process.
  - Allows connections to the Agent API object from both Policy Management and DMS clients.

  You can establish multiple connections to the Policy Server through the single Agent API object instance.
- A *user-defined* connection handle. You can create multiple user-defined connection objects; each one can support multiple connections to the Policy Server.

Establish a Default Connection

If you have not already established a connection to the Policy Server, you can request an automatic connection. If SiteMinder establishes a connection for you automatically, it creates a default Java Agent API object and handle. However, if a valid user-defined handle already exists, SiteMinder does not create a default object and handle. A user-defined handle takes precedence over a default handle.

To establish a default connection to the Policy Server automatically

1. Use the following constructor to create an API connection object:

```
SmApiConnection (boolean bDefaultAgentConnection
                                boolean disableLoadBalancing)
```

2. In the constructor, set bDefaultAgentConnection to true—for example:

```
SmApiConnection m_defaultConnection =
                                new SmApiConnection(true,false);
```

3. If bDefaultAgentConnection is false, you must explicitly establish the connection in your client code.

An automatic connection has the following requirements:

- Your Web Agent be installed on the same machine where you are running the Agent API.
- The property DefaultAgentName in the Web Agent configuration object contains an agent name. You define the Web Agent configuration object in the Policy Server.
- With Apache Web Agents, the path to the Agent configuration file is in the CLASSPATH. With Microsoft IIS Web Agents, this configuration information is in the Registry, so a CLASSPATH reference is not necessary.

Establish a User-Defined Connection

You establish a user-defined connection in one of two ways:

- By referencing an existing Java Agent API connection handle in the constructor of the SmApiConnection object.
- By establishing a new connection manually through the setAgentApiConnection() method.

Note: If you already have a connection to the Policy Server, you can use it to make subsequent Policy Management API or DMS API calls.

To create a connection using an existing Agent API connection

1. Create your connection object through the following constructor:

```
SmApiConnection (netegrity.siteminder.javaagent.AgentAPI
                                                agentApiConnection)
```

2. In the constructor, use agentApiConnection to pass in the handle of the existing Agent API connection—for example:

```
SmApiConnection myConnection =
                    new SmApiConnection (myAgentApiConnection);
```

   The new Java Agent API handle is a user-defined handle.

If you do not already have a default connection and you want a user-defined connection object, you can use the Agent API to create the agent object and then create the new connection, as follows:

1. Create the agent object.

   You can create an agent object based on connection parameters from either of the following sources:

   - User-defined connection parameters defined in your code—for example:

```
AgentAPI agent = new AgentAPI();
ServerDef sd = new ServerDef();
sd.serverIpAddress = POLICY_IP;
sd.connectionMin = CX_MIN;
sd.connectionMax = CX_MAX;
sd.connectionStep = CX_STEP;
sd.timeout = CX_TIMEOUT;
sd.authorizationPort = AZ_PORT;
sd.authenticationPort = AUTH_PORT;
sd.accountingPort = ACC_PORT;
InitDef init=new InitDef(AGENT_LOGIN,SHARED_SECRET,false, sd);
agent.init(init);
```

   Note: With SiteMinder v6.0 and later, the authorization, authentication, and accounting servers are combined into a single server process. Consequently, *authorizationPort*, *authenticationPort*, and *accountingPort* can all be set to the same value.

   - Connection parameters stored in an Agent configuration file.

   Note: SiteMider v4.x webagent.conf files are no longer supported by the SM API.

2. Create the new connection.

   After creating the agent object, you create the new connection in either of these ways:

   - Pass the agent object you just created into the constructor of the new SmApiConnection object—for example:

```
SmApiConnection myConnection = new SmApiConnection(agent);
```

   - Call setAgentApiConnection() and pass in the agent object you just created—for example:

```
SmApiConnection myConnection=new SmApiConnection(false,false);
myConnection.setAgentApiConnection(agent);
```

   If you establish the connection in this way, the Java Agent API handle is a user-defined handle.

   If you call setAgentApiConnection() and you do not have a connection, you can establish an automatic, static connection by passing in null.

Obtain a Session

After you obtain a connection to the Policy Server, get a user or administrator session.

Note: To use the Policy Management API, you must connect as a SiteMinder Administrator.

After you obtain a session object, pass it to the Policy Management API or the DMS API through the constructor for the SmPolicyApiImpl class, or the SmDmsApiImpl class.

To obtain a session, perform one of these actions:

If...	And...	Then...
You have an existing session from authenticating a user.	—	Pass in the session specification for the authenticated user.
You do not have an existing session.	You must connect as a SiteMinder Administrator.	Use the method SmApiSession.login().
You do not have an existing session.	You want to connect as a non-Administrator.	Use the Java Agent API to obtain a session specification for the user.

If you have a session specification for a user that has been authenticated, you can use that session specification. You need not obtain a new session specification.

To use an existing session, create an SmApiSession object and associate the session specification with that object.

Log in as a SiteMinder Administrator

To authenticate a SiteMinder administrator, use the login() method in the SmApiSession class of the Utilities package. This method uses the administrator's login credentials (username and password) to authenticate the administrator. Calling this login() method obtains a session specification and returns an SmApiResult object.

The syntax of the login() method is as follows:

```
result=mySession.login (username,
                        password,
                        IPaddress,
                        challengeReason);
```

Provide a value for the challengeReason parameter as follows:

- On the administrator's initial login, set challengeReason to 0 (no reason).
- If the initial login fails, use challengeReason in the next login() call to specify the results of the previous authentication attempt.

  To retrieve the reason value to assign to challengeReason, call getReason() in the SmApiResult object.

To obtain a new session specification for a user, use the Java Agent API to obtain a session specification. Then, create an SmApiSession object and associate the session specification with that object.

Make API Requests and Handle Results

After you establish a session, you can call the methods in your client application.

A *result* is a response from the Policy Server to a Java API request. Results are returned in an SmApiResult object.

Exceptions are thrown from an unexpected client-side error. An exception contains a result with additional information, such as the origin and severity of the result. To create a result object to store the results of API requests, use the constructor of the SmApiResult class in the Utilities package—for example:

```
SmApiResult result = new SmApiResult();
```

You can verify whether a request was successful by calling the method isSuccess() on the result object. The method returns true if the request was successful, or false if it was not successful.

You can compare the current result object to a specified result object by calling the equals() method.

You can use the equals() method to compare the current result object with SmApiResult constants that represent different kinds of results. For example, in the following code, the result represented by the unique constant SERVER_INVALID_PASSWORD is compared against the current result object:

```
InetAddress address = InetAddress.getLocalHost();
SmApiResult result = apiSession.login(usr,pwd,address,0);
boolean resultStatus =
            result.equals(SmApiResult.SERVER_INVALID_PASSWORD);
```

Log Trace Information

To log tracing information on the client side, use the -D option of the java tool and set the system property SMJAVASDK_LOG_INFO to true. SiteMinder logs the information to the standard output.

For example, if your Java Development Kit is on Windows and you want to trace the Policy Management API sample application, the command line would be:

```
java -DSMJAVASDK_LOG_INFO=true -classpath .;..\..\java\smjavasdk2.jar;
..\..\java\smjavaagentapi.jar PolicyApiSample
```

Javadoc Reference

This guide includes frequent references to the Javadoc, which you can access through a link on the SiteMinder bookshelf. Use the Javadoc to learn about a particular class or method. These details typically include syntax, parameters, return values, and exception information.

The description of each package, class, and interface in the Javadoc reference sometimes includes a Since heading that indicates the SiteMinder or SDK version when the component was introduced. Individual methods and fields only include a Since heading if they were added in a later version of the class or interface.

Support for Custom Code

CA supports the Software Development Kit (SDK) as part of the standard offerings. Code written by customers or partners, however, is not supported. You are responsible for the code you write. If you require assistance designing or implementing SDK-based code, contact your CA customer account team.

Chapter 2: Utilities Package

This section contains the following topics:

Purpose of the Utilities Package

If you plan to call functions in the Policy Management API or the DMS API, you must use the functions in the Utilities package to build your Java application.

The SmApiConnection, SmApiResult, SmApiSession, SmApiException, and SmProperty classes in the Utilities package provide services such as:

- Establishing a connection to the Policy Server
- Obtaining a session
- Providing a result object that stores results of API requests
- Handling exceptions and results
- Encapsulating property data

Classes for Internal Use

The Utilities package provides these classes for SiteMinder internal use only:

- SmApiConstants
- SmApiObject
- SmApiPropertySets
- SmApiExportFileHandler
- SmApiImportFileHandler
- SmFlag

These classes are used internally to define methods in the Policy Management API and the DMS API.

Connection Class

Use the SmApiConnection class to create an API connection object and establish a connection between the Agent API and Policy Server. Depending upon the constructor you use, you can establish either a default connection or a user-defined connection.

The core methods of the SmApiConnection class are as follows:

Method	Description
getAgentApiConnection()	Retrieves the Agent API handle for the current connection. Use this handle when issuing subsequent Java API requests to the Policy Server.
isValidApiConnection()	Verifies whether a valid Agent API connection is available.
setAgentApiConnection()	Establishes a user-defined connection through the handle passed into the method. If null is passed, a static connection is established.

Note: Do not call the execute() method from your client application. This method is for internal use only.

Session Class

The Session class, SmApiSession, lets you create a session object by passing in a valid API connection and, depending upon the constructor you use, a session specification. (A session specification is also known as a session ticket).

The core methods of the SmApiSession class are as follows:

Method	Description
getApiConnection()	Retrieves the SmApiConnection object for the current connection.
getSessionSpec()	Retrieves the specification for the current session.
login()	Logs in a SiteMinder administrator. The Policy Server issues a session specification for the session.
logout()	Logs out a SiteMinder administrator.

Method	Description
setApiConnection()	Sets a valid API connection.
setSessionSpec()	Sets an existing session specification.

Note: For login and logout of end users, DMS organization administrators, and DMS super administrators, use the AgentAPI.login() and AgentAPI.logout() methods in the Agent API package.

Result Class

The Result class, SmApiResult, stores the result of a SiteMinder Java API request. A SiteMinder result contains the following elements:

- Facility. Origin of the result. For example, the result might originate on the client or server.
- Severity. Significance of the result, such as informational or warning.
- Status. Status code of the result. Status codes are unique within each facility. You can use a facility's unique status code to distinguish a particular result from other results that might have originated from the facility.
- Message. Additional information about the result, such as descriptive text or numeric details.

A result might also include a reason code. For example, a password policy result might include a reason code of 1001, meaning that the password does not contain the required minimum number of characters. To find a reason code for a result, call getReason().

All server-side errors are returned as results, not as exceptions. However, when a client-side exception is thrown, an SmApiResult object is embedded in the exception.

Interpret a Result Object

Each result object and its Facility/Severity/Status combination are represented by a unique value. These unique values are associated with predefined constants defined in the SmApiResult class—for example, SERVER_CONFIGURATION_FAILURE.

To determine the Facility/Severity/Status information for a result, you can call the equals() method to compare the returned SmApiResult object with the result constants.

- Facility: FACILITY_CONNECTION
- Severity: SEVERITY_ERROR

- Status: 4
- Message: Unable to get server configuration

You can output a result object as a string—for example, you can generate a result string by calling toString() on the SmApiResult object.

A result string has five space-separated name/value pairs in the following format:

```
[facility=facility severity=severity reason=reason
status=statusCode message=message]
```

For example, suppose you call toString() for an SmApiResult object that occurs because a user attempted to create a password with fewer than the minimum number of alphabetic characters. The method might return a result string that looks like this:

```
[facility=4 severity=3 reason=1008 status=13 message=nArg=1,Arg1=3]
```

The fields in the result have the following meanings:

- facility=4. The result originated on the server.
- severity=3. The result is an error.
- reason=1008. The error occurred because the requested password has fewer than the minimum number of alphabetic characters required for passwords.
- status=13. The unique result status code for this facility.
- message=nArg=1,Arg1=3. An additional description of the result. The two parts to this field have the following meanings:
  - nArg1=1. The error contains just one error description.
  - Arg1=3. The error description is 3. In the context of reason code 1008, the Arg1 value means that a password is required to have a minimum of 3 alphabetic characters.

Core Methods in the Result Class

The core methods of the SmApiResult class include:

Method	Description
equals()	Indicates whether the current object is equal to the object passed to the method.
getError()	Retrieves a unique error code.
getFacility()	Retrieves the facility code associated with the error.

Method	Description
getMessage()	Retrieves the message associated with the error.
getReason()	Retrieves the reason code of the error.
getSeverity()	Retrieves the severity code associated with the error.
getStatus()	Retrieves the status code in the current facility. This method can take as a parameter the result code from the server.
isSuccess()	Reports whether the request was successful.
toString()	Returns a string representation of the SmApiResult object.

Exception Class

The Exception class, SmApiException, contains the result class SmApiResult. The following packages use SmApiException:

- com.netegrity.sdk.policyapi
- com.netegrity.sdk.dmsapi
- com.netegrity.sdk.apiutil

The core methods of the SmApiException class include:

Method	Description
getFacility()	Retrieves the facility code of the exception.
getReason()	Retrieves the reason code of the exception.
getSeverity()	Retrieves the severity code of the exception.
getStatus()	Retrieves the status code of the exception.
toString()	Returns a string representation of the SmApiResult object.

The Exception class extends java.lang.Exception. By calling the inherited getMessage() method, you can retrieve the message associated with the exception.

Property Class

The Property class, SmProperty, holds the following information about a property:

- Name
- Value
- Type (encrypted / plain)

The core methods of the SmProperty class include:

Method	Description
getName()	Retrieves the name of the property.
getType()	Retrieves the type of the property (that is, 0 if plain text, 1 if encrypted).
getValue()	Retrieves the value of the property.
setName()	Sets the name of the property.
setType()	Sets the type of the property, 0 if plain text & 1 if encrypted.
setValue()	Sets the value of the property.

Chapter 3: Agent API

This section contains the following topics:

SiteMinder Agents

A SiteMinder Agent is a client of the Agent API. The agent enforces access control policies served by the Policy Server. The Policy Server is a general-purpose policy engine with no specific knowledge of resources. The specific knowledge of resources is provided by SiteMinder agents. Agents establish resource semantics and act as gate keepers to protect resources from unauthorized users.

Different agent types protect different kinds of resources. Some agent types are pre-defined, standard agents that are shipped as part of the SiteMinder product—for example, the Web Agent, which provides HTTP access control for Web Servers. However, you can also use the Agent API to implement custom agents.

The Agent API lets you create a custom agent that can authenticate and authorize users in a variety of context-specific ways. For example, you could create an agent for FTP transfers that does the following:

- Implements certificate-based authentication instead of basic name and password credentials.
- Allows uploads and downloads based on an individual user's authorization level.

Custom agents can participate in a single sign-on environment with standard SiteMinder Web Agents.

Agent API Class Hierarchy

The primary point of access to the Java Agent API is the AgentAPI class. Several other classes are provided to hold data required by the AgentAPI class:

- Attribute
- AttributeList
- BinaryBuffer
- InitDef
- ManagementContextDef
- RealmDef
- ResourceContextDef
- ServerDef
- SessionDef
- TokenDescriptor
- TunnelServiceRequest
- UserCredentials

Implement the JNI Java Agent API

Applications that are built using the JNI Java AgentAPI either directly or indirectly (through another agent) are insulated from underlying implementation details, including:

- User namespaces, such as LDAP directories, SQL databases, or NT domains
- Authentication methods as simple as username/password or as complex as PKI systems
- Authorizations based on group membership or individual profile data

Additional benefits provided by the Java Agent API include full session management support, automatic encryption key rollover, and real-time policy updates.

To implement the JNI Java Agent API

1. Review the required software as listed in the accompanying release notes.
2. Review the sample code.

3. Write source code for your client application.
4. Ensure that your system can find the JNI support library when the Java Virtual Machine (JVM) is invoked, as follows:
   - On Windows: Change PATH to include the following, so that smjavaagentapi.dll can be found:

     <install_path>\sdk\bin

   - On Solaris: Change LD_LIBRARY_PATH to include the following, so that libsmjavaagentapi.so can be found:

     <install_path>/sdk/bin

   - On AIX: Change LIBPATH to include the following, so that libsmjavaagentapi.so can be found:

     <install_path>/sdk/bin

   - On Linux: Change LD_LIBRARY_PATH to include the following, so that libsmjavaagentapi.so can be found:

     <install_path>/sdk/bin

     Note: Java agents on Linux require Java SDK v 1.3.1.

   - On HP-UX 11: Change SHLIB_PATH to include the following, so that libsmjavaagentapi.sl can be found:

     <install_path>/sdk/bin

     Note: The Java Agent API is not available for HP10.
5. Ensure that SiteMinder can find the JNI Java AgentAPI JAR file when you compile or run an agent that uses the Java Agent API. The JAR file, smjavaagentapi.jar, is stored in the following locations:
   - Windows platforms:

     <install_path>\sdk\java

   - UNIX platforms:

     <install_path>/sdk/java

     Add smjavaagentapi.jar to your CLASSPATH setting. When compiling, you can use the -classpath switch.
6. Compile the Java Agent API application using javac.

   For an example, see java-build.bat or java-build.sh in the sample directory smjavaagentapi.
7. Configure the Policy Server to use the Java Agent API application.
8. Run the application.

   For an example, see java-run.bat or java-run.sh in the sample directory smjavaagentapi.

Implement the Pure Java Agent API

Applications that are built using the pure Java Agent API either directly or indirectly (through another agent) are insulated from underlying implementation details, including:

- User namespaces, such as LDAP directories, SQL databases, or NT domains
- Authentication methods as simple as username/password or as complex as PKI systems
- Authorizations based on group membership or individual profile data

Additional benefits provided by the Java Agent API include full session management support, automatic encryption key rollover, and real-time policy updates.

To implement the pure Java Agent API

1. Review the required software as listed in the accompanying release notes.
2. Review the sample code.
3. Write source code for your client application.
4. Ensure that SiteMinder can find the pure Java Agent API .jar file when you compile or run an agent that uses the Java Agent API. The JAR file, smagentapi.jar, is stored in the following locations:
   - Windows platforms:

     <install_path>\sdk\java
   - UNIX platforms:

     <install_path>/sdk/java

   Add smagentapi.jar to your CLASSPATH setting. When compiling, you can use the -classpath switch.
5. Compile the Java Agent API application using javac.

   For an example, see java-build.bat or java-build.sh in the sample directory smjavaagentapi.
6. Configure the Policy Server to use the Java Agent API application.
7. Run the application.

Pure Java Agent API Usage

Backward compatibility

The pure Java Agent API maintains binary and source compatibility with the JNI Java Agent API. The pure Java Agent API supports all of the other SiteMinder Java SDK interfaces that rely on the Agent API for connectivity to the SiteMinder Policy Server, including the SiteMinder Policy Management API and the SiteMinder DMS API, in addition to extending the portability of those interfaces.

Configuration limitations

The pure Java Agent API does not change the configuration of either the SiteMinder Application Server Agents or any agents developed with the SiteMinder SDK. The configuration of the pure Java Agent API is identical to the configuration of the JNI Java Agent API with the following exceptions:

- Migration of UNIX agents from the JNI Java Agent API to the pure Java Agent API requires re-registration of the trusted host entity with the SiteMinder Policy Server because the shared secret in the JNI Java API is computed differently from the pure Java implementation.
- On both Unix and Windows systems (due to file-locking incompatibilities with the native code Agent API), the SmHost.conf file cannot be shared between agents using the C/C++ or JNI Java Agent API and agents using the pure Java Agent API. Therefore, a separate copy of the bootstrap configuration file must be kept for pure Java Agent API agents.
- To register a host for a 5.x-type custom pure Java Agent you must use smreghost.bat (or smreghost.sh on UNIX), not smreghost.exe.
- Upgrades from the JNI Java Agent API on Unix systems requires users of custom 4.x-based agents that use shared secrets encrypted with the 4.x encryptkey tool to update their shared secret on the agent side for upgraded agents.

Connection to a Policy Server

Before an agent can perform work on behalf of its users, it must initialize connections to one or more Policy Servers by issuing the init() method. Through the InitDef parameter, you can specify connection parameters such as failover mode and connection pool size. This step creates TCP connections and typically does not need to be done more than once per agent instance.

After the Agent API is initialized, all API calls are fully thread-safe with respect to the initialized API instance.

It is possible to initialize more than one API instance (for example, when working with Policy Servers that use separate policy stores).

Immediately after initialization, the agent should communicate its version information to the Policy Server by calling doManagement() with the constant MANAGEMENT_SET_AGENT_INFO set in the ManagementContextDef object. The actual information can be any string containing enough information about the agent, such as the build number, and the version number. The string is recorded in the Policy Server logs.

After the Agent API has been initialized, the agent can perform useful work. At this point it can start accepting requests from its users, such as receiving GET requests for URLs.

User Access to Resources

The agent must perform the following steps before granting a user access to a requested resource. The outcome of most steps can be cached to improve agent performance. The agent can choose to cache as little or as much as possible.

1. Accept a user request.

   Accept a user request to access a resource. This is the application-specific request. For example, the Web Agent would accept a user's GET request for a URL.

2. Check if the resource is protected.

   Call isProtected() to determine if the requested resource is protected.

   If the resource is protected, the policy server returns the required credentials that must be obtained from the user in order to validate the user's identity. If the resource is not protected, access to the requested resource should be allowed.

   The outcome of this step can be cached.

3. Authenticate the user.

   Call login() to collect the required credentials from the user and to authenticate the user.

   Upon successful authentication, the Policy Server creates a session and returns response attributes, including the unique session id and session specification. These response attributes are policy-driven and may include user profile data, static or dynamic privileges, a number of predefined authentication state attributes, or any other data that was designated by a policy administrator.

   The agent can now perform session management by caching user session information and keeping track of session expiration.

4. Check if the user is authorized.

   Call authorize() to validate that the user has access to the requested resource.

   Upon successful authorization, the policy server returns response attributes including resource-specific privileges. These response attributes are policy driven and may include user profile data, static or dynamic privileges, or any other data that was designated by a policy administrator.

   At this point the user's authorization information with respect to the requested resource is known and can be cached to speed up future requests.

5. Audit cached authorization information.

   Both the authentication and authorization steps log the relevant information about the user, the protected resource, and the agent. However, if the agent performs authorizations out of its cache, the transaction can still be logged through the audit() method.

6. Allow access to resource.

   Now that the user's identity is known, authorization has been verified, and the required entitlements obtained, give the authorized user access to the resource.

7. Issue a management request.

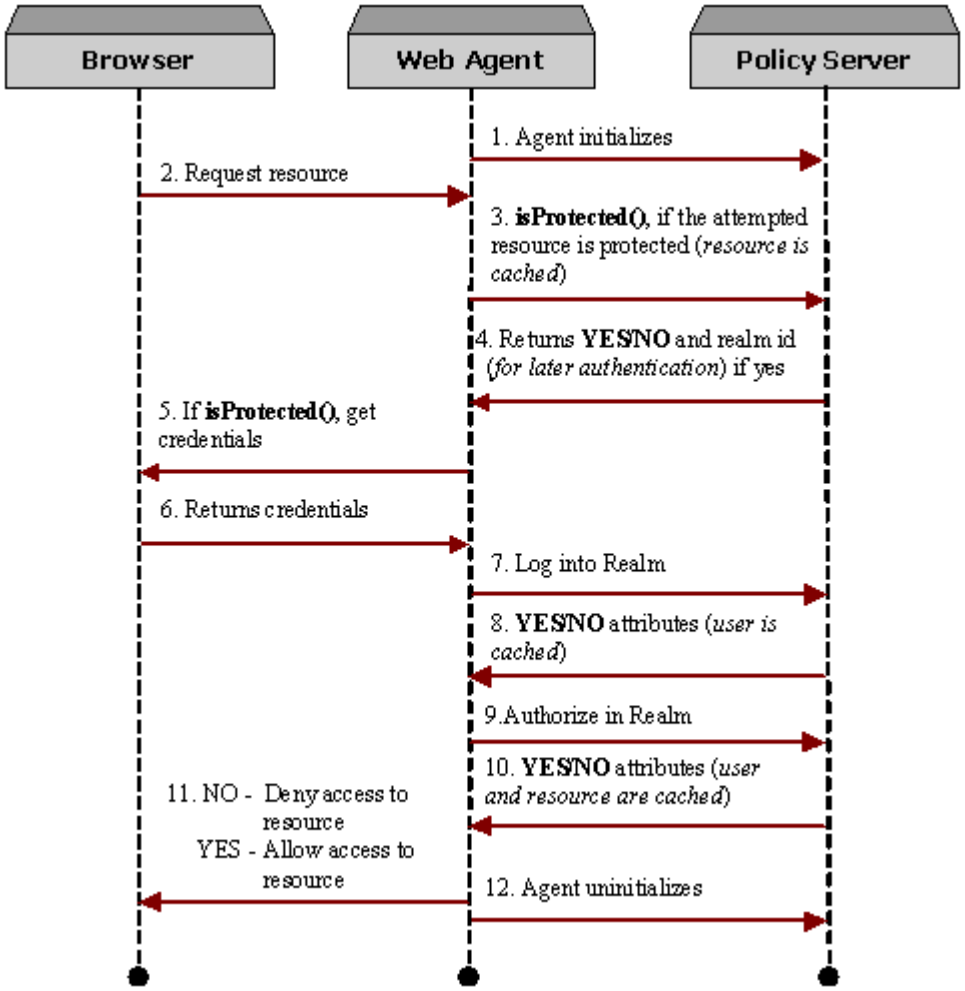
   This is an optional step that is used to poll the Policy Server for update commands. In response to a command, agents update encryption keys or flush caches or both.

When the agent is no longer needed, issue the unInit() method for each API instance. This closes TCP connections to all policy servers.

Note: The Agent API does not provide a facility for caching in a manner that enforces session validity. By choosing to cache user sessions and/or resource-specific privileges, the agent becomes obligated to perform its own session management during each user request. This is required, since caching on the agent removes the need to contact the SiteMinder Policy Server to perform session validation and/or resource authorizations.

How Web Agents Use the Agent API

The following figure shows the process flow that occurs when a Web Agent uses the Agent API:



Java Agent API Services

The Java Agent API provides a rich set of services that let you develop sophisticated, secure, and robust agents. Building an agent involves using these services:

- Session Services
- Authorization Services
- Auditing Services and Transaction Tracking
- Management Services (key encryption, cache updates)
- Tunnel Services

These services are accessed through the AgentAPI class.

Session Services

Sessioning is used to maintain consistent user sessions across multi-tiered application environments.

AgentAPI methods that implement session services are:

- login()
- logout()

Agents that perform session management use the sessioning services of the Java Agent API to create, delegate, validate, and terminate user sessions.

Note: For login and logout of SiteMinder administrators for Policy Server or DMS sessions, use the methods SmApiSession.login() and SmApiSession.logout() in the Utility package.

Session Creation and the Session Specification

A session is created after a successful user login. Once created, a user session persists until it is terminated.

When a user is authenticated, the Policy Server issues a *session specification*. A session specification contains information about the user.

User-side session persistence in a multi-tiered application environment is accomplished by saving and maintaining the user information in the session specification. This session specification represents a user session. It is the key to SiteMinder session management.

The SiteMinder environment where the user session was created is responsible for the creation, maintenance, and persistent storage of the session specification. For example, the Web Agent (HTTP environment) stores the session specification in an HTTP cookie.

Agents create sessions using login(). This method authenticates the user credentials and gets the information for session specification (including the unique session id). Once created, the session specification is updated on subsequent Java Agent API calls that also return updated expiration times. Agents can use this information to perform custom session management and keep track of session timeouts.

If your Web server's user-tracking feature is enabled, the SiteMinder Policy Server issues an *identity ticket* in addition to the session specification. Identity tickets can be used for identity-based personalization when a user is accessing a resource protected by anonymous authentication schemes. Identity tickets never expire.

Another important feature that is seamlessly integrated with the sessioning mechanism is the SiteMinder *universal ID*. A universal ID identifies the user to an application in a SiteMinder environment through a unique identifier, such as a social security number or customer account number. The universal ID facilitates identification of users between old and new applications by delivering the user's identification automatically, regardless of the application. Once configured on the Policy Server, a user's universal id becomes part of the session specification and is made available to agents for the duration of the entire session.

Session Validation

Agents request validation of a session specification to make sure that a user session has neither expired nor been terminated or revoked. This can occur at any time during the session's lifetime. Agents call AgentAPI.login() to validate a session specification.

Session Delegation

When an application's logic flow crosses application tiers, sessions may be delegated by passing the session specification between two agents. Each agent can choose to have the session specification validated.

Session Termination

A session is terminated in any of the following ways:

- After a user logs outs and the agent discards the session specification
- When the session expires
- When the session is revoked
- When the user account is disabled

To terminate a session, the agent must discard the session specification. Once a session is terminated, the user must log in again to establish a new session.

Authorization Services

Agents that perform access control functions use the authorization services of the AgentAPI class. These services enable clients to verify a user's rights to access a resource, retrieve a user's privileges with respect to specific resources, and determine the specific access control, if any, that is imposed upon a resource.

You can determine whether a resource is protected by calling the isProtected() method. This method accepts as a parameter the resource that is served by the requesting agent and returns information about the user's credentials.

Once the user's identity is validated, the agent calls the authorize() method to determine if the requesting user has access to the requested resource. Agents can perform fine-grained access control by leveraging the collection of response attributes that this method retrieves.

Auditing Services and Transaction Tracking

Agents can keep track of and log all user activity during a session. Although much of a user's activity is logged by the Policy Server, there are times when it may be necessary to log authorizations done out of agent cache. Agents call the audit() method to log such requests for resources.

By generating a unique transaction id, agents can correlate access control activity with application activity. The transaction id can be given to both the authorization and auditing methods so that the Policy Server would record the transaction-specific id associated with the application activity. This can be used for non-repudiation.

Management Services

A management protocol exists between agents and the SiteMinder Policy Server. This protocol helps an agent manage its caches and encryption keys in a manner consistent with both SiteMinder policies and administrative changes on the Policy Server.

To request the latest agent commands, an agent calls the method doManagement() with MANAGEMENT_GET_AGENT_COMMANDS set in the ManagementContextDef object. Typically, this call is made every *n* seconds by a thread running in the background. The types of agent commands that can be retrieved are cache commands and encryption commands.

Cache Commands

Cache commands inform the agent of any changes to its caches that may need to be made as a result of administrative updates to the Policy Server.

The cache commands are:

- CACHE_FLUSH_ALL
- CACHE_FLUSH_ALL_USERS
- CACHE_FLUSH_THIS_USER
- CACHE_FLUSH_ALL_REALMS
- CACHE_FLUSH_THIS_REALM

Encryption Commands

Encryption commands inform the agent of new encryption keys that are generated administratively or automatically by the Policy Server. Agents save secure state can use this protocol to keep track of the latest encryption keys.

The encryption commands are:

- AFFILIATE_KEY_UPDATE
- AGENT_KEY_UPDATE_NEXT
- AGENT_KEY_UPDATE_LAST
- AGENT_KEY_UPDATE_CURRENT
- AGENT_KEY_UPDATE_PERSISTENT

Tunnel Services

Tunnel services enable agents to establish secure communications with a callable service located on the Policy Server. This allows agents to perform custom actions over a secure, VPN-like channel without having to address issues such as encryption key management.

Response Attributes

Response attributes enable the Policy Server to deliver information to agents. Response attributes are managed through methods in the AgentAPI class.

There are two types of response attributes:

- Well-known
- Policy-based

The *well-known* attributes are always returned by the Policy Server after certain calls such as login(). These attributes represent static, fixed data such as the user DN and Universal ID.

The *policy-based* attributes are returned by the login() and authorize() methods. These attributes are based on policies and are the vehicle for delivering static and dynamic data from the Policy Server to agents, which can distinguish between authentication and authorization attributes. The actual source of the data is defined on the Policy Server using the responses feature that can be configured to deliver data from a variety of sources. Data may include static information, information from a directory profile, or a custom Policy Server plug-in. Once the responses are properly configured, agents are capable of performing fine-grained access control as well as profile-driven personalization.

Based on a policy definition, response attributes can time out or be cached for the duration of the user session. The Policy Server delivers an attribute along with the TTL (Time-To-Live) value, calculated in seconds. If the agent is caching user sessions and/or authorizations, it is responsible for keeping the relevant attributes up to date. Agents issue the updateAttributes() method to update stale attributes.

Single Sign-on

In a single sign-on environment, a user who successfully authenticates through a given agent does not have to re-authenticate when accessing a realm protected by a different agent. When a custom agent is involved in a single sign-on environment, the two agents must be in the same cookie domain—for example, xxx.domainname.com.

Single sign-on is made possible through a single sign-on cookie named SMSESSION. This cookie is created and written to the user's browser either by SiteMinder or by the custom agent.

Class AgentAPI contains two methods that allow custom agents to participate in a single sign-on environment with standard SiteMinder Web Agents:

decodeSSOToken()

The custom agent extracts the cookie's contents, called a token, from an existing SMSESSION cookie and passes the token to this method. The method decrypts the token and extracts the specified information. This method can also be used to update the last-access timestamp in the token.

createSSOToken()

After the user successfully logs in through the custom agent, the custom agent passes information about the user to this method. The method creates an encrypted token from this user information and from session information returned from the login call. The custom agent writes the token to the SMSESSION cookie.

See the sample custom agent code for an example of setting up the parameters for the single sign-on methods and parsing the results. The sample custom agent code is located in the smjavaagentapi directory of <install_path>\sdk\samples.

Log on through a Custom Agent

Here is the typical sequence of events in a single sign-on environment when the initial login is through the custom agent:

1. User logs in through the custom agent.
2. Custom agent calls login() to authenticate the user. The user is challenged for credentials.
3. Custom agent calls createSSOToken() and passes to it information about the user (user name, user DN, IP address of the requesting client). SiteMinder adds this information to a token along with session information returned from the login call. SiteMinder also encrypts the information in the token.
4. Custom agent creates the SMSESSION cookie in the user's browser and writes the token to the cookie.
5. User requests a resource protected by a standard SiteMinder agent.
6. The standard agent performs a login operation, which validates the user based on the information in the single sign-on cookie. The user is not challenged for credentials.

Log on through a Standard Agent

Here is the typical sequence of events that occurs in a single sign-on environment when the initial login is performed through the standard SiteMinder Web Agent:

1. User logs in through the standard agent.
2. Standard agent authenticates the user by challenging the user for credentials through the login call.
3. SiteMinder creates the SMSESSION cookie in the user's browser and inserts the encrypted token containing session information.
4. User requests a resource protected by a custom agent.
5. The custom agent obtains the SMSESSION cookie from the user's request and extracts the token.
6. The custom agent passes the token to the method decodeSSOToken(). The method decodes the token and returns a subset of the token's attributes to the custom agent.
7. The custom agent obtains the session specification from the token and passes the session specification to login(). The login call validates the user without challenging the user for credentials.
8. User requests a resource protected by a standard SiteMinder agent.
9. The standard agent performs a login operation, which validates the user based on the contents of the SMSESSION cookie. The user is not challenged for credentials.

Standard Agent Support

Custom agents created with the SiteMinder SDK v6.x can accept SMSESSION cookies created by a standard SiteMinder v4.x, v5.x, or v6.x Web Agent. However, standard SiteMinder v4.x or v5.x Web Agents can only accept cookies created by a custom agent if the standard agent has been upgraded with the appropriate Siteminder Agent Quarterly Maintenance Release (QMR). For information about the QMR version required for each standard agent version, see the accompanying SDK release notes.

To enable a SiteMinder v4.x or v5.x agent with the appropriate QMR upgrade to accept SMSESSION cookies created by a custom agent, the standard agent's Agent configuration file (LocalConfig.conf with IIS servers or WebAgent.conf with other servers) or central configuration object (for v5.x or higher) must contain the following entry:

```
AcceptTPCookie="yes"
```

Set AcceptTPCookie as follows:

- With 4.xQMR4 agents and above, add AcceptTPCookie="yes" directly in the standard agent's Agent configuration file.
- With 5.xQMR1 agents and above, add the entry to the standard agent's Agent Configuration Object if the AllowLocalConfig parameter for that object is set to no. If AllowLocalConfig is set to yes, you can set AcceptTPCookie in the standard agent's Agent configuration file.

How Information Is Bound to a Session

Session information can consist of more than the session specification. Session information can include any information that the client application wants to associate with the user's session.

Application-defined session information consists of name/value pairs called *session variables*. For example, business logic, certificate information, and SAML assertions for affiliate operations can all be stored as session variables and bound to the session ID.

The class AgentAPI provides the following methods for setting, retrieving, and deleting session variables:

- setSessionVariables()
- getSessionVariables()
- delSessionVariables()

Session variables are stored in a server-side database called the *session store*. The session store is managed by the Policy Server.

Advantages of Session Variables

When a client application uses session variables:

- Up to 4K of data can be stored for each session variable value.
- The session information persists across multiple Policy Servers. Centralizing session information on the server allows features such as cross-domain session management, including enforcing logout and idle timeout across different domains.

Requirements for Using Session Variables

For a client application to use session variables, both of the following prerequisites must be met:

- The Session Server must be enabled in the Policy Server Management Console.
- During realm configuration in the Policy Server UI, Persistent Session must be selected for at least one of the realms to be accessed during the session. As soon as the user accesses a realm configured for persistent sessions, session variables can be used throughout the remainder of the session.

End of Session Cleanup

When the user logs out and the agent discards the session specification, the session ends. In the case of a persistent session, SiteMinder removes all session information, including any session variables, from the session store.

Server Clusters

To help prevent service interruptions, SiteMinder includes a failover feature. If the primary Policy Server fails and failover is enabled, a backup Policy Server takes over policy operations. Beginning with SiteMinder v6.0, failover can occur not only between Policy Servers, but between groups, or *clusters*, of Policy Servers.

The cluster functionality also improves server performance by providing dynamic load balancing between the servers in a cluster. With dynamic load balancing, policy operations are automatically distributed between the available servers in a cluster according to the performance capabilities of each server.

Clustered and Non-Clustered Servers

An agent running against Agent API v6.x can be associated with one or more Policy Servers, or with one or more clusters of Policy Servers, as follows:

- **Clustered servers**

  In the ServerDef object for each clustered server, set clusterSeq() to the sequence number for the cluster. All servers in a cluster have the same cluster sequence number.

  Behavior: Failover occurs between clusters of servers if multiple clusters are defined. Also, requests to servers within a cluster are sent according to the improved performance-based load-balancing techniques introduced with Agent API v6.0.

- **Non-clustered servers**

  In the ServerDef object for each non-clustered server, set the method clusterSeq() to 0.

  Behavior: Behavior is the same as in v5.x installations—that is, you can enable failover among the servers associated with the agent, or you can enable round-robin behavior among the servers.

  When round-robin behavior is enabled, the improved performance-based load-balancing techniques introduced with Agent API v6.0 are used.

Note: The same agent cannot be associated with both clustered and non-clustered servers.

Cluster Configuration

You can configure a cluster through the Agent API or through a host configuration object using the Policy Server User Interface. If you configure a cluster through the Agent API, be sure that the configuration does not conflict with any cluster configuration information that may be defined in the host configuration object.

You configure the individual servers or clusters of servers that the agent is associated with through the InitDef and ServerDef classes.

Cluster failover occurs according to the following configuration settings:

- **Failover threshold**. The minimum percentage of servers within a cluster that must be available for Policy Server requests. If the number of available servers falls below the threshold, failover to the next cluster occurs.

  The failover threshold percentage applies to all clusters associated with the agent.

  To determine the number of servers that the percentage represents in any given cluster, multiply the threshold percentage by the number of servers in the cluster, rounding to the nearest integer. For example, with a 60-percent failover threshold for a cluster of five servers, failover to the next cluster occurs when the number of available servers in the currently active cluster falls below 3.

  Set through: InitDef constructor that contains the failOverThreshold parameter.

- **Sequence of cluster failover**. Each cluster is assigned a sequence number. When cluster failover occurs, SiteMinder sends subsequent Policy Server requests to the next cluster in the cluster sequence.

  Set through: ServerDef.clusterSeq().

Cluster Failover

If your site uses clusters, you typically will have a primary cluster and one or more backup clusters.

The *primary cluster* is the cluster to which SiteMinder sends requests as long as the number of available servers in the cluster does not fall below the failover threshold. If there are not enough available servers in the primary cluster, failover to the next cluster in the cluster sequence occurs. If that cluster also fails, failover to the third cluster occurs, and so on.

When All Clusters Fail

If the number of available servers falls below the failover threshold in all clusters associated with the agent, policy operations do not stop. Requests are sent to the first cluster in the cluster sequence that has at least one available server.

For example, suppose an agent is associated with two clusters—C1 containing three servers, and C2 containing five servers.The failover threshold for any cluster associated with the agent is set at 60 percent.

The following table shows the minimum number of servers that must be available within each cluster:

Cluster	Servers in Cluster	Percentage Failover Threshold	Numeric Failover Threshold (Minimum Available Servers)
C1	3	60	1
C2	5	60	3

If the number of available servers falls below the threshold in each cluster, so that C1 has no available servers and C2 has just two, the next incoming request will be dispatched to a C2 server with the best response time. After at least two of the three C1 servers are repaired, subsequent requests are load-balanced among the available C1 servers.

Version Compatibility and Failover Behavior

Agent API v6 is backwards-compatible with Agent API v5, allowing complete interoperability between v5/v6 agents and the v5/v6 Agent APIs.

Timeouts

Agents can enforce session timeouts or rely on the Policy Server to validate each request. Typically, caching of user sessions or privileges by the agent requires some form of timeout enforcement on the agent side. In this case, the agent is responsible for keeping track of last access time and knowing when the session is going to expire.

Agents that do not cache can leverage the Policy Server's enforcement of timeouts. The following Java Agent API methods return the updated timeout information after every call:

- login()
- authorize()
- audit()

Agent Type

The Agent Type defines the behavior of an agent. After you have developed a custom agent, you must configure a new Agent Type for the agent in the Policy Server User Interface. For example, if you developed a custom FTP Agent, you would then need to configure an Agent Type for the FTP Agent in the Policy Server User Interface.

Note: For information on configuring an Agent Type for your custom agent, see the *SiteMinder Programming Guide for C*.

Chapter 4: Policy Management API

This section contains the following topics:

About Policy Management

Policy management consists of creating, deleting, and modifying policy objects within a SiteMinder policy store. Through the Policy Management API, you can perform most of the data manipulations that you can perform through the native Policy Server User Interface. For example, you can write a client application that allows administrators to perform tasks such as:

- Creating a policy domain
- Creating an Agent object
- Creating an Agent configuration object
- Creating a host configuration object
- Registering a trusted host
- Creating a SiteMinder user directory object
- Creating an authentication scheme object
- Creating an administrator
- Creating a realm
- Adding a realm to a policy domain
- Creating a rule
- Creating a response
- Creating a policy
- Adding a user or group to a policy
- Adding a rule to a policy
- Setting responses for rules in a policy
- Migrating an entire policy store or an individual policy domain remotely

Policy Management Setup

To run applications built with the Policy Management API:

- Use the Policy Server Management Console to configure the Policy Server so that it points to the policy store you want to access.
- Run your Policy Management application on the same machine as the Policy Server or on a machine that has network access to the Policy Server.

Note: If an application built with the Policy Management API runs on the same machine as the Policy Server, the application must run as the same user who installed the Policy Server (for example, smuser on UNIX platforms).

Required JAR File

The JAR file smjavasdk2.jar is required for building and running Policy Management applications. The JAR file is stored in the following locations:

- Windows platforms:

  <install_path>\sdk\java

- UNIX platforms:

  <install_path>/sdk/java

Policy Store Objects

Interface SmPolicyApi is implemented by the class SmPolicyApiImpl. Use this class as the starting point for the Policy Management API. Each policy store object is associated with a class in the Policy Management API. You create and manage policy store objects through the methods in an object's class.

Policy store objects can be classified according to scope:

- *Domain objects are* visible only within the domain. They cannot be shared between domains.
- *Global object*s are visible across all domains.

  Global objects are sometimes called system objects.

Global objects include:

- Administrators
- Agent types
- Agents and agent groups

- Agent Configuration objects
- Host Configuration objects
- Trusted Hosts
- Authentication schemes
- Authentication/authorization maps
- Certificate maps
- Domains
- ODBC query schemes
- Password policies
- Registration schemes
- User directories

Domain objects include:

- Policies
- Realms
- Responses and response groups
- Response attributes
- Rules and rule groups
- User policies

When you are working in the Policy Server user interface, you will see most of the above objects listed in the System and Domain tabs of the SiteMinder Administration window.

Note: Descriptions in the Javadoc reference specify whether an object has global scope or domain scope.

Write a Policy Management Application

To write a Policy Management application

1. Establish a Connection to the Policy Server
2. Obtain a Session Object
3. Pass in the Session Object
4. Make Policy Management API Requests
5. Terminate the Administrator Session

The SiteMinder SDK contains a sample of how to use the classes and methods in the Java Policy Management API.

Establish a Connection to the Policy Server

To establish a connection to the Policy Server, use the SmApiConnection class of the Utilities package. This class holds the Agent API handle through which Java API requests are sent.

There are two types of connection handles in this class:

- A *default* connection handle. A default connection handle:
  - Represents a single instance of an Agent API object.
  - Is static across the process.
  - Allows connections to the Agent API object from both Policy Management and DMS clients.

  You can establish multiple connections to the Policy Server through the single Agent API object instance.
- A *user-defined* connection handle. You can create multiple user-defined connection objects; each one can support multiple connections to the Policy Server.

Obtain a Session Object

A session object is obtained when a user or administrator successfully logs in. In this case, an administrator login is required, since only administrators can perform policy management.

To log in a SiteMinder administrator and establish an administrator session, call the login() method in the SmApiSession class of the Utilities package.

Once login is successful, the session object will hold a valid administrator session specification.

Pass in the Session Object

After obtaining a valid session, create a Policy Management API object by passing the session to the constructor of the SmPolicyApiImpl class—for example:

```
SmPolicyApi policyApi = new SmPolicyApiImpl (apiSession);
```

In the example, policyApi is the new Policy Management API object and *apiSession* is the session obtained when the administrator successfully logged in.

Make Policy Management API Requests

After you obtain a session object and create a Policy Management API object, you are ready to make Policy Management requests. Most of the methods in the Policy Management API are categorized according to the SiteMinder object that a given method acts upon—for example, agents, policies, and rules.

There is also a Utilities category for methods that perform services, such as cache and encryption key management. Use these categories to help you find a particular Policy Management API method to use in your custom policy management applications.

Note: The methods in the policyapi package can only be called from a Siteminder administrator session.

Terminate the Administrator Session

When you are finished making Policy Management API requests, log out the administrator by calling the logout() method in the SmApiSession class of the Utilities package.

Administrator Methods

Unless otherwise specified, the following methods are in the class SmPolicyApiImpl. The following methods act on administrator objects. You create an administrator object by instantiating SmAdmin.

Method	Description
addAdmin()	Adds an administrator object to the policy store.
addAdminToDomain()	Associates an administrator with a domain.
deleteAdmin()	Deletes an administrator.
getAdmin()	Gets the contents of an administrator.
getAdminUserDirs()	Gets a list of user directories that an administrator can manage.
modifyAdmin()	Modifies an administrator.
removeAdminFromDomain()	Disassociates an administrator from a domain.

Agent Methods

Unless otherwise specified, the methods listed in this table are in the class SmPolicyApiImpl. The following methods act on agent objects. You create an agent object by instantiating SmAgent.

Method	Description
addAgent()	Adds an agent object to the policy store.
deleteAgent()	Deletes an agent.
getAgent()	Gets the contents of an agent.
modifyAgent()	Modifies an agent.

Agent Configuration Object Methods

Unless otherwise specified, the methods listed in this table are in the class SmPolicyApiImpl. The following methods act on agent configuration objects. You define an agent configuration object by instantiating SmAgentConfig.

Method	Description
addAgentConfig()	Adds an agent configuration object to the policy store.
deleteAgentConfig()	Deletes an agent configuration object.
getAgentConfig()	Gets the contents of an agent configuration object.
modifyAgentConfig()	Modifies an agent configuration object.

Authentication and Authorization Map Methods

Unless otherwise specified, the methods listed in this table are in the class SmPolicyApiImpl. The following methods act on authentication and authorization directory mapping objects. You create an authentication and authorization directory mapping object by instantiating SmAuthAzMap.

Method	Description
addAuthAzMap()	Adds an authentication and authorization directory mapping object to the policy store.
deleteAuthAzMap()	Deletes an authentication and authorization directory mapping object.
getAuthAzMap()	Gets the contents of an authentication and authorization directory mapping object.
modifyAuthAzMap()	Modifies an authentication and authorization directory mapping object.

Authentication Scheme Methods

Unless otherwise specified, the methods listed in this table are in the class SmPolicyApiImpl. The following methods act on authentication schemes. You create an authentication scheme by instantiating SmScheme.

Method	Description
addScheme()	Adds an authentication scheme to the policy store.
deleteScheme()	Deletes an authentication scheme.
getScheme()	Gets the contents of an authentication scheme.
modifyScheme()	Modifies an authentication scheme.

Certificate Map Methods

Unless otherwise specified, the methods listed in this table are in the class SmPolicyApiImpl. The following methods act on certificate mapping objects. You create certificate mapping objects by instantiating SmCertMap.

Method	Description
addCertMap()	Adds a certificate mapping object to the policy store.
deleteCertMap()	Deletes a certificate mapping object.
getCertMap()	Gets the contents of a certificate mapping object.
modifyCertMap()	Modifies a certificate mapping object.

Domain Methods

Unless otherwise specified, the methods listed in this table are in the class SmPolicyApiImpl. The following methods act on domain objects. You create domain objects by instantiating SmDomain.

Method	Description
addDomain()	Adds a domain object to the policy store.

Method	Description
deleteDomain()	Deletes a domain.
getDomain()	Gets the contents of a domain.
getDomainObject()	Gets a domain object for the specified object name or OID.
getDomainObjectNames()	Gets a list of domain objects within a domain.
isDomainObject()	Indicates whether an object is a domain object. In classes SmObjectImpl, SmDomainObjectImpl.
modifyDomain()	Modifies a domain.

General Object Methods

Unless otherwise specified, the methods listed in this table are in the class SmPolicyApiImpl. The following methods act on multiple types of objects.

Method	Description
getGlobalObjectNames()	Gets a list of global objects.
getObject()	Gets a global object for the specified object name or OID.
getOid()	Retrieves an object identifier for an object. In class SmObjectImpl.
isWriteable()	Specifies whether an object is writeable. In classes SmAgentType, SmDomainObjectImpl, and SmObjectImpl.
renameObject()	Renames an object.

Group Methods

Unless otherwise specified, the methods listed in this table are in the class SmPolicyApiImpl. The following methods act on group objects. Group objects are created with SmAgentGroup (for agent groups), SmResponseGroup (for response groups), or SmRuleGroup (for rule groups).

Method	Description
addGroup()	Adds an agent, response, or rule group to the policy store.
addToGroup()	Adds a group element of type rule, response, or agent to the specified group.
deleteGroup()	Deletes an existing group.
getGroup()	Gets the contents of an existing group.
getGroupMembers()	Get a list of groups of all types.
modifyGroup()	Modify a group.
removeFromGroup()	Removes a group element from a group.

Host Configuration Object Methods

Unless otherwise specified, the methods listed in this table are in the class SmPolicyApiImpl. The following methods act on host configuration objects. You define a host configuration object by instantiating SmHostConfig.

Method	Description
addHostConfig()	Adds a host configuration object to the policy store.
deleteHostConfig()	Deletes a host configuration object.
getHostConfig()	Gets the contents of a host configuration object.
modifyHostConfig()	Modifies a host configuration object.

ODBC Query Scheme Methods

Unless otherwise specified, the methods listed in this table are in the class SmPolicyApiImpl. The following methods act on ODBC Query schemes. You create ODBC Query schemes by instantiating SmODBCQuery.

Method	Description
addODBCQuery()	Adds an ODBC query object to the policy store.
deleteODBCQuery()	Deletes an ODBC query object.
getODBCQuery()	Gets the contents of an ODBC query object.
modifyODBCQuery()	Modifies an ODBC query object.

Password Policy Methods

Unless otherwise specified, the methods listed in this table are in the class SmPolicyApiImpl. The following methods act on password policy objects. You create password policy objects by instantiating SmPasswordPolicy.

Method	Description
addPasswordPolicy()	Adds a password policy object to the policy store.
deletePasswordPolicy()	Deletes a password policy.
getPasswordPolicy()	Gets the contents of a password policy.
isEnabled()	Specifies whether the password policy is enabled. In class SmPasswordPolicy.
isEntireDir()	Specifies whether the password policy applies to the entire directory. In class SmPasswordPolicy.
modifyPasswordPolicy()	Modifies a password policy.

Policy Methods

The following methods act on policy and policy link objects. A policy link is an association of a policy, a rule, and optionally, a response. Unless otherwise specified, these methods are in the class SmPolicyApiImpl.

Policy objects are created with SmPolicy. Policy link objects are created with SmPolicyLink.

Method	Description
addPolicy()	Adds a policy object to the policy store.
addPolicyLink()	Adds a policy link to a policy.
deletePolicy()	Deletes the policy associated with the specified domain.
deletePolicyLink()	Removes a policy link from a policy.
getPolicy()	Gets the contents of a policy.
getPolicyLinks()	Gets all of the policy links for the specified policy and domain.
modifyPolicy()	Modify the policy associated with the specified domain.
modifyPolicyLink()	Modifies the specified policy link.

Policy Migration Methods

The following methods enable you to migrate policy data between remote Policy Servers. Unless otherwise specified, these methods are in the class SmPolicyApiImpl.

Functionally, the remote policy data export and import methods behave in the same manner as the smobjexport and smobjimport utilities.

Policy export attributes are set with SmExportAttr. Policy import parameters are set with SmImportAttr.

Method	Description
doExport()	Exports an entire policy store or a single policy domain from a remote Policy Server and writes the output on the client's local file system.

Method	Description
doImport()	Imports an entire policy store or a single policy domain onto a remote Policy Server.

Realm Methods

The following methods act on realm objects. Realm objects are created with SmRealm.

Method	Description
addRealm(()	Adds a realm object to the policy store.
deleteRealm()	Deletes a realm.
getRealm()	Gets the contents of a realm.
getRealmRules()	Gets all the rules for the specified realm and domain.
getRealmUserPolicies()	Gets a list of user policies that can access a realm.
modifyRealm()	Modifies the specified realm.

Response Methods

The following methods act on response and response attribute objects. Unless otherwise specified, these methods are in the class SmPolicyApiImpl. Response objects are created with SmResponse. Response attribute objects are created with SmResponseAttr.

Method	Description
addResponse()	Adds a response object to the policy store.
addResponseAttr()	Creates a response attribute and associates it with a response.
deleteResponse()	Deletes a response.
deleteResponseAttribute()	Deletes a response attribute.
getResponse()	Gets the contents of a response.

Method	Description
getResponseAttrs()	Gets a list of attributes for the specified response.
modifyResponse()	Modify the specified response.
setResponseInPolicyLink()	Changes the response for the specified policy link.

Root Configuration Methods

The following methods act on root configuration objects. Unless otherwise specified, the methods listed in this table are in the class SmPolicyApiImpl. You create root configuration objects by instantiating SmRootConfig.

Method	Description
addRootConfig()	Adds a root configuration object to the policy store.
deleteRootConfig()	Deletes a root configuration.
getRootConfig()	Gets the contents of a root configuration.
modifyRootConfig()	Modifies a root configuration.

Rule Methods

The following methods act on rule objects. Unless otherwise specified, the methods listed in this table are in the class SmPolicyApiImpl. You create rule objects by instantiating SmRule.

Method	Description
addRule()	Adds a rule object to the policy store.
deleteRule()	Deletes a rule.
getRule()	Gets the contents of a rule.
modifyRule()	Modifies a rule.

Self-Registration Methods

The following methods act on self-registration objects. Unless otherwise specified, the methods listed in this table are in the class SmPolicyApiImpl. You create self-registration objects by instantiating SmSelfReg.

Method	Description
addSelfReg()	Adds a self-registration object to the policy store.
deleteSelfReg()	Deletes a self-registration object.
getSelfReg()	Gets the contents of a self-registration object.
modifySelfReg()	Modifies a self-registration object.

Trusted Host Object Methods

The following methods act on Trusted Host objects. Unless otherwise specified, the methods listed in this table are in the class SmPolicyApiImpl. You define a Trusted Host object by instantiating SmTrustedHost.

Method	Description
addTrustedHost()	Registers a trusted host with the Policy Server.
deleteTrustedHost()	Deletes a trusted host object.

User Directory Methods

User management functionality is provided in the DMS API. However, the Policy Management API provides methods for getting and setting user attributes. These methods are in the SmUserDirectory class.

For example:

- To specify which user attribute holds the disabled state of the user, call setDisabledAttr() in SmUserDirectory.
- To disable and enable users, use the DMS API.

The following methods act on user directory objects. Unless otherwise specified, the methods listed in this section are in the class SmPolicyApiImpl. You create user directory objects by instantiating SmUserDirectory.

Method	Description
addUserDirectory()	Adds a user directory object to the policy store.
addUserDirToDomain()	Associates an existing user directory with a domain.
deleteUserDirectory()	Deletes a user directory.
getDirectoryContents()	Gets a list of distinguished names and classes for the specified user directory.
getUserDirectory()	Gets the contents of a user directory.
getUserDirSearchOrder()	Retrieves the search order of user directories for a domain by retrieving a vector of user directory names.
lookupDirectory()	Gets a list of distinguished names and classes for the specified user directory and search pattern.
modifyUserDirectory()	Modifies a user directory.
removeUserDirFromDomain()	Disassociates an existing user directory from a domain.
setUserDirSearchOrder()	Sets the search order of user directories in a domain.

User Policy Methods

The following methods act on user policy objects. Unless otherwise specified, the methods listed in this table are in the class SmPolicyApiImpl. You create user policy objects by instantiating SmUserPolicy.

Method	Description
addUserPolicy()	Adds a user policy object to the policy store.

Method	Description
deleteUserPolicy()	Deletes a user policy for a specified domain.
getUserPolicies()	Gets all the user policies for the specified policy and domain.

Utility Methods

The following methods provide a variety of services, including cache and encryption key management. Unless otherwise specified, the methods listed in this table are in the class SmPolicyApiImpl.

Method	Description
changeDynamicKey()	Changes a dynamic encryption key.
changePersistentKey()	Changes the persistent encryption key.
changeSessionKey()	Changes the session encryption key.
flushAll()	Flushes all SiteMinder caches.
flushRealm()	Flushes a realm from the resource cache.
flushRealms()	Flushes all realms from the resource cache.
flushUser()	Flushes a user from the user information cache.
flushUsers()	Flushes all users from the information cache.
search()	Searches the specified object.
setApiSession()	Sets the API session object.

Object Associations

Some objects can be associated with or disassociated from one another—for example, AddAdminToDomain() adds an administrator object to a domain, and RemoveAdminFromDomain() removes an administrator object from a domain. An add-to operation requires that both objects exist prior to the call. After a remove-from operation, both objects still exist, but they are no longer associated with one other.

When you are looking for a method that associates or disassociates two objects, look in the category of the method that you are adding or removing. For example, AddAdminToDomain() and RemoveAdminFromDomain() are both found in Administrator Methods.

Add Objects to the Policy Store

After creating a Policy Management API object, you can create objects to add to the policy store.

To add objects to the policy store

1. Create an object to be added to the policy store.

   For example, if you want to create an agent object:

   ```
   SmAgent agent = new SmAgent();
   ```

2. Set the appropriate fields for the object—for example:

   ```
   agent.setName ("myAgent");
   agent.setSecret ("siteminder");
   agent.setDescription ("Sample agent");
   agent.setAgentType (SmAgentType.DefaultAgentType);
   ```

3. Add the object to the policy store, as follows:

   - Call the add... method for the object you just created—for example, addAgent() for an agent object, or addDomain() for a domain object—and pass in the object you want to add to the policy store.
   - Returning the result into a result object.

   For example:

   ```
   result = policyApi.addAgent(agent);
   ```

4. Examine the result.

If the call is successful:

- The method returns an SmApiResult object whose isSuccess() method returns true.
- The object is added to the SiteMinder policy store.
- The *Oid* field in the corresponding object structure is set to the object identifier.

Retrieve Objects from the Policy Store

To retrieve an object from the policy store

1. Create an object of the relevant class to store the returned properties. For example, the following code creates an agent object:

   ```
   SmAgent myAgent = new SmAgent();
   ```

2. Call the appropriate get... function for the object you just created—for example, getAgent() for an agent object, or getDomain() for a domain object—and pass in the object you just created. For example, if you're retrieving an agent named myAgent:

   ```
   result = myPolicyApi.getAgent ("myAgent", myAgent);
   ```

If the method succeeds, it populates *myAgent* with the properties of the specified agent object. (If a get... method retrieves a list, the list is written to a vector.) If no matching objects are found, the properties of the receiving object retain their initial values.

Delete Objects from the Policy Store

A delete operation deletes an object from the policy store. You can only delete one object at a time from the policy store.

To delete an object, use the object-deletion method for the object you're deleting—for example, deleteAgent() for an agent object, or deleteDomain() for a domain object

Authentication Scheme Configuration

When you configure an authentication scheme programmatically, you provide information that would otherwise be provided through the Authentication Scheme Properties dialog box of the Policy Server UI.

When you configure an authentication scheme, you use the get... and set... methods in the SmScheme class to provide the following information:

- Scheme type

  SiteMinder provides a number of standard authentication scheme types (also called templates). Each authentication scheme type is configured differently. The scheme types are descibed in subsequent topics.

- Description

  Brief description of the authentication scheme.

- Protection level

  Protection level values can range from 1 through 1000. The higher the number, the greater the degree of protection provided by the scheme.

- Library

  An authentication scheme library performs authentication processing for the associated authentication scheme type. Each pre-defined authentication scheme is shipped with a default library, which you typically will use. But optionally, you can use a custom library instead of the default.

- Parameter

  Additional information that the authentication scheme requires, such as the URL of an HTML login page.

  With some authentication schemes, the parameter information is constructed from field values in the Scheme Type Setup tab of the Authentication Scheme Properties dialog box. To see how a parameter string might be constructed for a given scheme type, open this dialog box, choose the appropriate scheme type, provide values to the fields in the Scheme Type Setup tab, and view the constructed parameter in the Advanced tab.

  For information on providing parameter values for different authentication scheme types, see the chapter on authentication schemes in the *Policy Design Guide*.

- Shared Secret

  Information that is known to both the authentication scheme and the Policy Server. Different authentication schemes use different kinds of secrets. Most schemes use no secret.

- Is template?

  A flag that specifies whether the authentication scheme is a template.

- Is used by administrator?

  A flag that specifies whether the authentication scheme can be used to authenticate administrators.

- Save Credentials?

  A flag that specifies whether the user's credentials will be saved.

- Is RADIUS?

  A flag that specifies whether the scheme can be used with RADIUS agents.

- Ignore password check?

  A flag that specifies whether password policies for the scheme are enabled. If True (1), password policies will be disabled.

Note: These categories of information can be used for different purposes in different authentication schemes. For example, with the TeleID authentication scheme, the shared secret is used to supply the encryption seed.

Configuration Information

When you configure an authentication scheme, you provide the following kinds of configuration information. This information is managed through get... and set... methods in the SmScheme class.

Note: The categories of information below can be used for different purposes in different authentication schemes. For example, with the TeleID authentication scheme, the shared secret is used to supply the encryption seed.

- Scheme type

  SiteMinder provides a number of standard authentication scheme types (also called templates). Each authentication scheme type is configured differently.

  The following scheme types are available. The are listed below as they appear in the Authentication Scheme Type field of the Policy Server Authentication Scheme Properties dialog box:

  - Anonymous Template
  - Basic Over SSL Template
  - Basic Template
  - CRYPTOCard RB-1 Template
  - Custom Template
  - HTML Form Template
  - Impersonation Template
  - MS Passport Template
  - RADIUS CHAP/PAP Template
  - RADIUS Server Template
  - SafeWord HTML Form Template
  - SafeWord Template
  - SecurID HTML Form Template (ACE server)
  - SecurID Template (ACE server)
  - smauthetsso Authentication Scheme (uses Custom Template)
  - TeleID Template (Encotone)
  - Windows Authentication Template
  - X.509 Client Cert and Basic Template
  - X.509 Client Cert and Form Template
  - X.509 Client Cert or Basic Template
  - X.509 Client Cert or Form Template

- X.509 Client Cert Template

- Description

  Brief description of the authentication scheme.

- Protection level

  Protection level values can range from 1 through 1000. The higher the number, the greater the degree of protection provided by the scheme.

- Library

  An authentication scheme library performs authentication processing for the associated authentication scheme type. Each pre-defined authentication scheme is shipped with a default library, which you typically will use. But optionally, you can use a custom library instead of the default.

- Parameter

  Additional information that the authentication scheme requires, such as the URL of an HTML login page.

  With some authentication schemes, the parameter information is constructed from field values in the Scheme Type Setup tab of the Authentication Scheme Properties dialog box. To see how a parameter string might be constructed for a given scheme type, open this dialog box, choose the appropriate scheme type, provide values to the fields in the Scheme Type Setup tab, and view the constructed parameter in the Advanced tab.

- Shared Secret

  Information that is known to both the authentication scheme and the Policy Server. Different authentication schemes use different kinds of secrets. Most schemes use no secret.

- Is template?

  A flag that specifies whether the authentication scheme is a template.

- Is used by administrator?

  A flag that specifies whether the authentication scheme can be used to authenticate administrators.

- Save Credentials?

  A flag that specifies whether the user's credentials will be saved.

- Is RADIUS?

  A flag that specifies whether the scheme can be used with RADIUS agents.

- Ignore password check?

  A flag that specifies whether password policies for the scheme are enabled. If True (1), password policies will be disabled.

Anonymous Template

Use this table when configuring an authentication scheme based on the scheme type Anonymous. The Java methods referenced in the table are in the class SmScheme.

Information Type	Value Assignment and Meaning
Scheme type	setType(TypeAnonymous) The scheme type Anonymous.
Description	setDescription(*description*) The description of the authentication scheme.
Protection level	setLevel(0) Set to 0. Not applicable to this scheme type.
Library	setLibrary("smauthanon") The default library for this scheme type.
Parameter	setParameter(*param*) A string containing the guest DN. Policies associated with the guest DN must apply to anonymous users.
Shared secret	setSecret("") Set to an empty string. Not applicable to this scheme.
Is template?	setIsTemplate(0) Set to false (0) to indicate that the scheme is not a template.
Is used by administrator?	setIsUsedByAdmin(0) Set to false (0)—scheme is not used to authenticate administrators.
Save credentials?	setAllowSaveCreds(0) Set to false (0) to indicate that user credentials won't be saved.
Is RADIUS?	setIsRadius(0) Set to false (0)—scheme is not used with RADIUS agents.
Ignore password check?	setIgnorePwCheck(1) Set to true (1)—ignore password checking.

Basic Template

Use this table when configuring an authentication scheme based on the scheme type Basic. The Java methods referenced in the table are in the class SmScheme.

Information Type	Value Assignment and Meaning
Scheme type	setType(TypeBasic) The scheme type Basic.
Description	setDescription(*description*) The description of the authentication scheme.
Protection level	setLevel(*value*) A value of 1 through 1000. The higher the number, the greater degree of protection provided by the scheme. Default is 5.
Library	setLibrary("smauthdir") The default library for this scheme type.
Parameter	setParameter("") Set to an empty string. Not applicable to this scheme.
Shared secret	setSecret("") Set to an empty string. Not applicable to this scheme.
Is template?	setIsTemplate(0) Set to false (0) to indicate that the scheme is not a template.
Is used by administrator?	setIsUsedByAdmin(1) Set to true (1)—scheme can be used to authenticate administrators.
Save credentials?	setAllowSaveCreds(0) Set to false (0) to indicate that user credentials won't be saved.
Is RADIUS?	setIsRadius(1) Set to true (1)—scheme can be used with RADIUS agents.
Ignore password check?	setIgnorePwCheck(*flag*) Set to true (1) to ignore password checking, or false (0) to check passwords. Default is 0.

Basic Over SSL Template

Use this table when configuring an authentication scheme based on the scheme type Basic over SSL. The Java methods referenced in the table are in the class SmScheme.

Information Type	Value Assignment and Meaning
Scheme type	setType(TypeBasicOverSSL) The scheme type Basic over SSL.
Description	setDescription(*description*) The description of the authentication scheme.
Protection level	setLevel(*value*) A value of 1 through 1000. The higher the number, the greater degree of protection provided by the scheme. Default is 10.
Library	setLibrary("smauthcert") The default library for this scheme type.
Parameter	setParameter(*param*) A string containing the domain or IP address of the SSL server and the name of the SSL Credentials Collector (SCC). Format: https://*server*/*SCC*?basic The following example uses the default SCC: https://my.server.com/siteminderagent/nocert/smgetcred.scc?basic
Shared secret	setSecret("") Set to an empty string. Not applicable to this scheme.
Is template?	setIsTemplate(0) Set to false (0) to indicate that the scheme is not a template.
Is used by administrator?	setIsUsedByAdmin(0) Set to false (0) for this scheme.
Save credentials?	setAllowSaveCreds(0) Set to false (0) to indicate that user credentials won't be saved.
Is RADIUS?	setIsRadius(0) Set to false (0)—scheme is not used with RADIUS agents.
Ignore password check?	setIgnorePwCheck(*flag*) Set to true (1) to ignore password checking, or false (0) to check passwords. Default is 0.

CRYPTOCard RB-1 Template

Use this table when configuring an authentication scheme based on the scheme type CRYPTOCard RB-1. The Java methods referenced in the table are in the class SmScheme.

Information Type	Value Assignment and Meaning
Scheme type	setType(TypeCryptoCard) The scheme type CRYPTOCard RB-1.
Description	setDescription(*description*) The description of the authentication scheme.
Protection level	setLevel(*value*) A value of 1 through 1000. The higher the number, the greater degree of protection provided by the scheme. Default is 15.
Library	setLibrary("smauthcryptocard") The default library for this scheme type.
Parameter	setParameter("") Set to an empty string. Not applicable to this scheme.
Shared secret	setSecret(*secret*) The secret used to initialize the token cards. The secret is stored in the CRYPTOCard Ccsecret file. This file is supplied by the token vendor.
Is template?	setIsTemplate(0) Set to false (0) to indicate that the scheme is not a template.
Is used by administrator?	setIsUsedByAdmin(1) Set to true (1)—scheme can be used to authenticate administrators.
Save credentials?	setAllowSaveCreds(0) Set to false (0) to indicate that user credentials won't be saved.
Is RADIUS?	setIsRadius(1) Set to true (1)—scheme can be used with RADIUS agents.
Ignore password check?	setIgnorePwCheck(1) Set to true (1)—ignore password checking.

Custom Template

Use this table when configuring an authentication scheme based on the scheme type Custom. You create custom schemes using the C Authentication API. For more information, see the *Developer's Guide for C*. The Java methods referenced in the table are in the class SmScheme.

Information Type	Value Assignment and Meaning
Scheme type	setType(TypeCustom) The scheme type Custom.
Description	setDescription(*description*) The description of the authentication scheme.
Protection level	setLevel(*value*) A value of 0 through 1000. The higher the number, the greater degree of protection provided by the scheme. Default is 5.
Library	setLibrary(*customLibName*) The name of the custom shared library you created using the C Authentication API.
Parameter	setParameter(*param*) Any string of one or more parameters required by your custom authentication scheme. For a custom authentication scheme that uses SSL, you must supply a URL that points to a SiteMinder Web Agent library required for the SSL-based authentication.
Shared secret	setSecret(*secret*) The shared secret, if any, that your custom authentication scheme uses for encryption of credentials.
Is template?	setIsTemplate(0) Set to false (0) to indicate that the scheme is not a template.
Is used by administrator?	setIsUsedByAdmin(*flag*) Set to true (1) to specify that the scheme can be used to authenticate administrators, or to false (0) to specify that the scheme cannot be used to authenticate administrators. Default is 0.
Save credentials?	setAllowSaveCreds(0) Set to false (0) to indicate that user credentials won't be saved.

Information Type	Value Assignment and Meaning
Is RADIUS?	setIsRadius(0) Set to false (0)—scheme is not used with RADIUS agents.
Ignore password check?	setIgnorePwCheck(*flag*) Set to true (1) to ignore password checking, or false (0) to check passwords. Default is 0.

HTML Form Template

Use this table when configuring an authentication scheme based on the scheme type HTML Form. The Java methods referenced in the table are in the class SmScheme.

Information Type	Value Assignment and Meaning
Scheme type	setType(TypeHTMLForm) The scheme type HTML Form.
Description	setDescription(*description*) The description of the authentication scheme.
Protection level	setLevel(*value*) A value of 1 through 1000. The higher the number, the greater degree of protection provided by the scheme. Default is 5.
Library	setLibrary("smauthhtml") The default library for this scheme type.
Parameter	setParameter(*param*) A string containing a user attribute list plus the location of the forms credential collector (FCC). The attribute list must begin with AL= and use commas as the list delimiter character, and it must end with a semicolon—for example: AL=Password,SSN,age,zipcode; The complete parameter format is: *attr-list*;https:/*server/fcc* The following example uses the default FCC: AL=PASSWORD,SSN,age,zipcode; http://my.server.com/siteminderagent/forms/login.fcc
Shared secret	setSecret("") Set to an empty string. Not applicable to this scheme.

Information Type	Value Assignment and Meaning
Is template?	setIsTemplate(0) Set to false (0) to indicate that the scheme is not a template.
Is used by administrator?	setIsUsedByAdmin(0) Set to false (0)—scheme is not used to authenticate administrators.
Save credentials?	setAllowSaveCreds(*flag*) Set to true (1) to indicate that user credentials should be saved, or false (0) to indicate that user credentials should not be saved. Default is 0.
Is RADIUS?	setIsRadius(0) Set to false (0)—scheme is not used with RADIUS agents.
Ignore password check?	setIgnorePwCheck(*flag*) Set to true (1) to ignore password checking, or false (0) to check passwords. Default is 0.

Impersonation Template

Use this table when configuring an authentication scheme based on scheme type Impersonation. The Java methods referenced in the table are in the class SmScheme.

Information Type	Value Assignment and Meaning
Scheme type	setType(TypeImpersonation) The scheme type Impersonation.
Description	setDescription(*description*) The description of the authentication scheme.
Protection level	setLevel(*value*) A value of 1 through 1000. The higher the number, the greater degree of protection provided by the scheme. Default is 5.
Library	setLibrary("smauthimpersonate") The default library for this scheme type.

Information Type	Value Assignment and Meaning
Parameter	setParameter(*param*) A string containing a user attribute list plus the location of the forms credential collector (FCC). The attribute list must begin with AL= and use commas as the list delimiter character, and it must end with a semicolon—for example: AL=Password,SSN,age,zipcode; The complete parameter format is: *attr-list*;https:/*server/fcc* The following example uses the default FCC: AL=PASSWORD,SSN,age,zipcode; http://my.server.com/siteminderagent/ forms/imp.fcc
Shared secret	setSecret("") Set to an empty string. Not applicable to this scheme.
Is template?	setIsTemplate(*templateFlag*) Set to false (0) to indicate that the scheme is not a template.
Is used by administrator?	setIsUsedByAdmin(0) Set to false (0)—scheme is not used to authenticate administrators.
Save credentials?	setAllowSaveCreds(0) Set to false (0) to indicate that user credentials won't be saved.
Is RADIUS?	setIsRadius(0) Set to false (0)—scheme is not used with RADIUS agents.
Ignore password check?	setIgnorePwCheck(1) Set to true (1)—ignore password checking.

MS Passport Template

Use this table when configuring an authentication scheme based on scheme type MS Passport. The Java methods referenced in the table are in the class SmScheme.

Information Type	Value Assignment and Meaning
Scheme type	setType(TypeMSPassport) The scheme type MS Passport.

Information Type	Value Assignment and Meaning
Description	setDescription(*description*) The description of the authentication scheme.
Protection level	setLevel(*value*) A value of 1 through 1000. The higher the number, the greater degree of protection provided by the scheme. Default is 1.
Library	setLibrary("smauthmspp") The default library for this scheme type.
Parameter	setParameter(*param*) The following information, separated by semicolons: ■ A DN for an anonymous user. Format: anonuser=*anonUserDN* If you specify an anonymous user DN, the protection level is 0. ■ The search string for looking up a user in a user directory of the specified type. Format: attribute=*nameSpace*:*attrib*=*searchSpec* Valid namespaces are LDAP, AD, ODBC, WinNT, and Custom. ■ The registration URL. The URL can be a custom URL or a SiteMinder form. Formats: registrationurl=*URL* (custom URL) registrationurl=FORM=*URL* (SiteMinder form) Example using an LDAP attribute and a custom URL: attribute=LDAP:altSecurityIdentities= Kerberos:%s@company.local;registrationurl =http://passport.xanadu.local/registration/passportreg.asp
Shared secret	setSecret("") Set to an empty string. Not applicable to this scheme.
Is template?	setIsTemplate(*templateFlag*) Set to false (0) to indicate that the scheme is not a template.
Is used by administrator?	setIsUsedByAdmin(0) Set to false (0)—scheme is not used to authenticate administrators.
Save credentials?	setAllowSaveCreds(0) Set to false (0) to indicate that user credentials won't be saved.

Information Type	Value Assignment and Meaning
Is RADIUS?	setIsRadius(0) Set to false (0)—scheme is not used with RADIUS agents.
Ignore password check?	setIgnorePwCheck(1) Set to true (1)—ignore password checking.

RADIUS CHAP/PAP Template

Use this table when configuring an authentication scheme based on the scheme type RADIUS CHAP/PAP. The Java methods referenced in the table are in the class SmScheme.

Information Type	Value Assignment and Meaning
Scheme type	setType(TypeRadiusChapPap) The scheme type RADIUS CHAP/PAP.
Description	setDescription(*description*) The description of the authentication scheme.
Protection level	setLevel(*value*) A value of 1 through 1000. The higher the number, the greater degree of protection provided by the scheme. Default is 5.
Library	setLibrary("smauthchap") The default library for this scheme type.
Parameter	setParameter(*param*) A string containing the name of a user directory attribute. This attribute is used as the clear text password for authentication.
Shared secret	setSecret("") Set to an empty string. Not applicable to this scheme.
Is template?	setIsTemplate(0) Set to false (0) to indicate that the scheme is not a template.
Is used by administrator?	setIsUsedByAdmin(0) Set to false (0)—scheme is not used to authenticate administrators.
Save credentials?	setAllowSaveCreds(0) Set to false (0) to indicate that user credentials won't be saved.

Information Type	Value Assignment and Meaning
Is RADIUS?	setIsRadius(1) Set to true (1)—scheme can be used with RADIUS agents.
Ignore password check?	setIgnorePwCheck(*flag*) Set to true (1) to ignore password checking, or false (0) to check passwords. Default is 0.

RADIUS Server Template

Use this table when configuring an authentication scheme based on the scheme type RADIUS Server. The Java methods referenced in the table are in the class SmScheme.

Information Type	Value Assignment and Meaning
Scheme type	setType(TypeRadiusServer) The scheme type RADIUS Server.
Description	setDescription(*description*) The description of the authentication scheme.
Protection level	setLevel(*value*) A value of 1 through 1000. The higher the number, the greater degree of protection provided by the scheme. Default is 5.
Library	setLibrary("smauthradius") The default library for this scheme type.
Parameter	setParameter(*param*) A string containing the IP address and port of the RADIUS server—for example: 123.123.12.12:1645 The default UDP port is 1645.
Shared secret	setSecret(*secret*) The user attribute that the RADIUS Server will use as the clear text password.
Is template?	setIsTemplate(0) Set to false (0) to indicate that the scheme is not a template.
Is used by administrator?	setIsUsedByAdmin(1) Set to true (1)—scheme can be used to authenticate administrators.

Information Type	Value Assignment and Meaning
Save credentials?	setAllowSaveCreds(0) Set to false (0) to indicate that user credentials won't be saved.
Is RADIUS?	setIsRadius(1) Set to true (1)—scheme can be used with RADIUS agents..
Ignore password check?	setIgnorePwCheck(*flag*) Set to true (1) to ignore password checking, or false (0) to check passwords. Default is 0.

SafeWord HTML Form Template

Use this table when configuring an authentication scheme based on the scheme type SafeWord HTML Form. The Java methods referenced in the table are in the class SmScheme.

Information Type	Value Assignment and Meaning
Scheme type	setType(TypeSafeWordHTMLForm) The scheme type SafeWord HTML Form.
Description	setDescription(*description*) The description of the authentication scheme.
Protection level	setLevel(*value*) A value of 1 through 1000. The higher the number, the greater degree of protection provided by the scheme. Default is 10.
Library	setLibrary("smauthenigmahtml") The default library for this scheme type.
Parameter	setParameter(*param*) A string containing the name and location of the forms credentials collector. This example shows the default credentials collector: http://my.server.com/siteminderagent/forms/safeword.fcc
Shared secret	setSecret("") Set to an empty string. Not applicable to this scheme.
Is template?	setIsTemplate(0) Set to false (0) to indicate that the scheme is not a template.

Information Type	Value Assignment and Meaning
Is used by administrator?	setIsUsedByAdmin(1) Set to true (1)—scheme can be used to authenticate administrators.
Save credentials?	setAllowSaveCreds(0) Set to false (0) to indicate that user credentials won't be saved.
Is RADIUS?	setIsRadius(1) Set to true (1)—scheme can be used with RADIUS agents..
Ignore password check?	setIgnorePwCheck(1) Set to true (1)—ignore password checking.

SafeWord Template

Use this table when configuring an authentication scheme based on the scheme type SafeWord. The Java methods referenced in the table are in the class SmScheme.

Information Type	Value Assignment and Meaning
Scheme type	setType(TypeSafeWordServer) The scheme type SafeWord.
Description	setDescription(*description*) The description of the authentication scheme.
Protection level	setLevel(*value*) A value of 1 through 1000. The higher the number, the greater degree of protection provided by the scheme. Default is 10.
Library	setLibrary("smauthenigma") The default library for this scheme type.
Parameter	setParameter("") Set to an empty string. Not applicable to this scheme.
Shared secret	setSecret("") Set to an empty string. Not applicable to this scheme.
Is template?	setIsTemplate(0) Set to false (0) to indicate that the scheme is not a template.

Information Type	Value Assignment and Meaning
Is used by administrator?	setIsUsedByAdmin(1) Set to true (1)—scheme can be used to authenticate administrators.
Save credentials?	setAllowSaveCreds(0) Set to false (0) to indicate that user credentials won't be saved.
Is RADIUS?	setIsRadius(1) Set to true (1)—scheme can be used with RADIUS agents..
Ignore password check?	setIgnorePwCheck(1) Set to true (1)—ignore password checking.

SAML Artifact Template

Use this table when configuring an authentication scheme based on SiteMinder Federation Security Services. The Federation Security Services feature is licensed separately. The Java methods referenced in the table are in the class SmScheme.

Information Type	Value Assignment and Meaning
Scheme type	setType(TypeSAMLArtifact) The scheme type SAML Artifact.
Description	setDescription(*description*) The description of the authentication scheme.
Protection level	setLevel(*value*) A value of 1 through 1000. The higher the number, the greater degree of protection provided by the scheme. Default is 5.
Library	setLibrary("smauthsaml") The default library for this scheme type.

Information Type	Value Assignment and Meaning
Parameter	setParameter(*param*) The following required parameters: ■ Name. The name of the affiliate. ■ RedirectMode. The way in which the SAML Credentials Collector redirects to the target resource. One of the following numeric values: 0. Meaning: 302 No Data. 1. Meaning: 302 Cookie Data. 2. Meaning: Server Redirect. ■ SRCID. The 20-byte source ID for the site that produces the SAML assertion. The ID is located at the SAML producer's site in the properties file AMAssertionGenerator.properties. ■ AssertionRetrievalURL. The URL for obtaining the assertion from the SAML assertion producer's site. ■ Audience. The URI of the document that describes the agreement between the portal and the affiliate. This value is compared with the audience value specified in the SAML assertion. ■ Issuer. The SAML issuer specified in the assertion. ■ AttributeXPath. A standard XPath query run against the SAML assertion. The query obtains the data that is substituted in a search specification that looks up a user. ■ attribute. The search string for looking up a user in a user directory of the specified type. Use a percent sign (%) to indicate where the value returned from the XPath query should be inserted. For example, if you specify attribute **LDAP:uid=%s**, and **user1** is returned from the query, the search string used for LDAP directories is **uid=user1**. At least one attribute must be specified. Format of the parameter string is as follows. Separate name/value pairs with semi-colons (;). The format example includes LDAP and ODBC attributes: Name=*name*;RedirectMode=0|1|2;SRCID=*srcid*; AssertionRetrievalURL=*url*;Audience=*audience*; Issuer=*issuer*;AttributeXpath=*XPathQuery*; attribute=LDAP:*srch*Spc;attribute=ODBC:*srchSpc*
Shared secret	setSecret(*secret*) The password for the affiliate site.

Information Type	Value Assignment and Meaning
Is template?	setIsTemplate(*templateFlag*) Set to false (0) to indicate that the scheme is not a template.
Is used by administrator?	setIsUsedByAdmin(0) Set to false (0)—scheme is not used to authenticate administrators.
Save credentials?	setAllowSaveCreds(0) Set to false (0) to indicate that user credentials won't be saved.
Is RADIUS?	setIsRadius(0) Set to false (0)—scheme is not used with RADIUS agents.
Ignore password check?	setIgnorePwCheck(1) Set to true (1)—ignore password checking.

SecurID HTML Form Template

Use this table when configuring an authentication scheme based on the scheme type SecurID HTML Form. The Java methods referenced in the table are in the class SmScheme.

Information Type	Value Assignment and Meaning
Scheme type	setType(TypeACEServerHTMLForm) The scheme type SecurID HTML Form.
Description	setDescription(*description*) The description of the authentication scheme.
Protection level	setLevel(*value*) A value of 1 through 1000. The higher the number, the greater degree of protection provided by the scheme. Default is 15.
Library	setLibrary("smauthacehtml") The default library for this scheme type.

Information Type	Value Assignment and Meaning
Parameter	setParameter(*param*) A string containing the name of the attribute that contains the ACE IDs, the Web server where the forms credential collector (FCC) is installed, and the target executable file required for processing SecurID authentication with forms support. It also specifies whether an SSL connection is required. Format: *attr*;https://*server*/*target* **Note:** The "s" in "https" is optional, depending on whether you want an SSL connection. The following example uses the default for processing SecurID authentication with forms support: ace_id;https://my.server.com/siteminderagent/pwcgi/smpwservicescgi.exe
Shared secret	setSecret("") Set to an empty string. Not applicable to this scheme.
Is template?	setIsTemplate(0) Set to false (0) to indicate that the scheme is not a template.
Is used by administrator?	setIsUsedByAdmin(0) Set to false (0)—scheme is not used to authenticate administrators.
Save credentials?	setAllowSaveCreds(0) Set to false (0) to indicate that user credentials won't be saved.
Is RADIUS?	setIsRadius(0) Set to false (0)—scheme is not used with RADIUS agents.
Ignore password check?	setIgnorePwCheck(1) Set to true (1)—ignore password checking.

SecurID Template

Use this table when configuring an authentication scheme based on the scheme type SecurID. The Java methods referenced in the table are in the class SmScheme.

Information Type	Value Assignment and Meaning
Scheme type	setType(TypeACEServer) The scheme type SecurID.
Description	setDescription(*description*) The description of the authentication scheme.
Protection level	setLevel(*value*) A value of 1 through 1000. The higher the number, the greater degree of protection provided by the scheme. Default is 15.
Library	setLibrary("smauthace") The default library for this scheme type.
Parameter	setParameter(*param*) A string containing the attribute in the authentication user directory that contains the ACE Server user ID.
Shared secret	setSecret("") Set to an empty string. Not applicable to this scheme.
Is template?	setIsTemplate(0) Set to false (0) to indicate that the scheme is not a template.
Is used by administrator?	setIsUsedByAdmin(1) Set to true (1)—scheme can be used to authenticate administrators.
Save credentials?	setAllowSaveCreds(0) Set to false (0) to indicate that user credentials won't be saved.
Is RADIUS?	setIsRadius(1) Set to true (1)—scheme can be used with RADIUS agents.
Ignore password check?	setIgnorePwCheck(1) Set to true (1)—ignore password checking.

smauthetsso Authentication Scheme

The smauthetsso authentication scheme is similar to the SiteMinder X.509 certification scheme, but with an eSSO cookie as the authentication credential instead of an X.509 credential.

If this scheme is configured for either cookieorbasic or cookieorforms mode, and both an eSSO cookie and login name and password credentials are passed to it, the eSSO cookie is ignored, and the login name and password are used to authenticate the user to SiteMinder.

When the eSSO cookie is the only credential, the authentication scheme uses the ETWAS API to connect to the configured eSSO Policy Server to validate the cookie and extract the user Distinguished Name (DN) from it.

Use this table when configuring an smauthetsso authentication scheme, which is based on the scheme type Custom. The Java methods referenced in the table are in the class SmScheme.

Information Type	Value Assignment and Meaning
Scheme type	setType(TypeCustom) Uses the scheme type Custom.
Description	setDescription(*description*) The description of the authentication scheme.
Protection level	setLevel(*value*) A value of 0 through 1000. The higher the number, the greater degree of protection provided by the scheme. Default is 5.
Library	setLibrary("smauthetsso") The name of the library for this authentication scheme.

Information Type	Value Assignment and Meaning
Parameter	setParameter(*param*) An ordered set of tokens, separated by semi-colons: <*Mode*>[; <*Target*>]; <*Admin*>; <*eTPS_Host*> You can add spaces to make the string easier to read. <*Mode*> specifies the type of credentials that the authenticaion scheme will accept. The following values are possible: ■ cookie -- Only SSO Cookies are acceptable. ■ cookieorbasic -- If an SSO Cookie is not provided, a login name and password are requested by using Basic Authentication. ■ cookieorforms -- If an SSO Cookie is not provided, a login name and password are requested by using Forms Authentication. <*Target*> is valid only with cookieorforms mode. This is identical to the Target field for standard HTML Forms Authentication Scheme. <*Admin*> specifies the login ID of an administrator for the Policy Server. The password for this administrator has been specified in the Shared Secret field. <*eTPO_Host*> specifies the name of the amchine on which the Policy Server is installed. SiteMinder will authenticate itself as <*Admin*> to the Policy Server on the <*eTPS_Host*> so that SiteMinder can request validation of SSO cookies. Examples: "cookie; SMPS_sso; myserver.myco.com" "cookieorforms; /siteminderagent/forms/login.fcc; SMPS_sso; myserver.myco.com"
Shared secret	setSecret(*secret*) The password of the Policy Server administrator named in the Parameter field.
Is template?	setIsTemplate(0) Set to false (0) to indicate that the scheme is not a template.

Information Type	Value Assignment and Meaning
Is used by administrator?	setIsUsedByAdmin(*flag*) Set to true (1) to specify that the scheme can be used to authenticate administrators, or to false (0) to specify that the scheme cannot be used to authenticate administrators. Default is 0.
Save credentials?	setAllowSaveCreds(0) Set to false (0) to indicate that user credentials won't be saved.
Is RADIUS?	setIsRadius(0) Set to false (0)—scheme is not used with RADIUS agents.
Ignore password check?	setIgnorePwCheck(*flag*) Set to true (1) to ignore password checking, or false (0) to check passwords. Default is 0.

TeleID Template

Use this table when configuring an authentication scheme based on the scheme type TeleID. The Java methods referenced in the table are in the class SmScheme.

Information Type	Value Assignment and Meaning
Scheme type	setType(TypeEncotone) The scheme type TeleID.
Description	setDescription(*description*) The description of the authentication scheme.
Protection level	setLevel(*value*) A value of 1 through 1000. The higher the number, the greater degree of protection provided by the scheme. Default is 15.
Library	setLibrary("smauthencotone") The default library for this scheme type.
Parameter	setParameter("") Set to an empty string. Not applicable to this scheme.
Shared secret	setSecret(*seed*) The encryption seed. SiteMinder uses this value as an encryption seed for initializing hardware tokens.
Is template?	setIsTemplate(0) Set to false (0) to indicate that the scheme is not a template.

Information Type	Value Assignment and Meaning
Is used by administrator?	setIsUsedByAdmin(1) Set to true (1)—scheme can be used to authenticate administrators.
Save credentials?	setAllowSaveCreds(0) Set to false (0) to indicate that user credentials won't be saved.
Is RADIUS?	setIsRadius(1) Set to true (1)—scheme can be used with RADIUS agents..
Ignore password check?	setIgnorePwCheck(1) Set to true (1)—ignore password checking.

Windows Authentication Template

Use this table when configuring an Integrated Windows Authentication scheme based on the scheme type Windows Authentication (previously known as NTLM). This scheme type is used to authenticate against WinNT or Active Directory user stores.

An Active Directory can be configured to run in *mixed mode* or *native mode*. An Active Directory supports WinNT style authentication when running in mixed mode. In native mode, an Active Directory supports only LDAP style lookups.

This authentication scheme supports either mixed mode or native mode.

The Java methods referenced in the table are in the class SmScheme.

Information Type	Value Assignment and Meaning
Scheme type	setType(TypeNTLM) The scheme type Windows Authentication (NTLM).
Description	setDescription(*description*) The description of the authentication scheme.
Protection level	setLevel(*value*) A value of 1 through 1000. The higher the number, the greater degree of protection provided by the scheme. Default is 5.
Library	setLibrary("smauthntlm") The default library for this scheme type.

Information Type	Value Assignment and Meaning
Parameter	setParameter(*param*) The value of *param* determines the style of authentication to perform for this scheme: **NTLM authentication** (for WinNT or Active Directory running in mixed mode) Format: *iis-web-server-url/path-to-ntc-file* In the format, *iis-web-server-url* is the name of the IIS web server that is the target of the redirection, and *path-to-ntc-file* is the location of the .ntc file that collects the WinNT credentials. For example: http://myiiswebserver.mycompany.com/ siteminderagent/ntlm/creds.ntc A SiteMinder Web Agent must be installed on the specified server. By default, the Web Agent installation creates a virtual directory for NTLM credential collection. **Windows Authentication (for Active Directory running in native mode)** With this authentication style, *param* has an LDAP filter added to the beginning of the redirection URL. The filter and URL are separated by a semi-colon (;). For example: cn=%{UID},ou=Users,ou=USA,dc=%{DOMAIN}, dc=mycompany,dc=com;http:// myiiswebserver.mycompany.com/ siteminderagent/ntlm/creds.ntc SiteMinder uses the LDAP filter to map credentials received from the browser/Web Agent to an LDAP DN or search filter.
Shared secret	setSecret("") Set to an empty string. Not applicable to this scheme.
Is template?	setIsTemplate(0) Set to false (0) to indicate that the scheme is not a template.
Is used by administrator?	setIsUsedByAdmin(0) Set to false (0)—scheme is not used to authenticate administrators.
Save credentials?	setAllowSaveCreds(0) Set to false (0) to indicate that user credentials will not be saved.

Information Type	Value Assignment and Meaning
Is RADIUS?	setIsRadius(0) Set to false (0)—scheme is not used with RADIUS agents.
Ignore password check?	setIgnorePwCheck(*flag*) For WinNT and for Active Directory running in mixed mode, this property must be true (1)—ignore password checking. For Active Directory running in native mode, set to true (1) to ignore password checking, or false (0) to check passwords. The default is 0.

X.509 Client Cert and Basic Template

Use this table when configuring an authentication scheme based on the scheme type X.509 Client Certificate *and* Basic. The Java methods referenced in the table are in the class SmScheme.

Information Type	Value Assignment and Meaning
Scheme type	setType(TypeX509ClientCertAndBasic) The scheme type X.509 Client Certificate and Basic.
Description	setDescription(*description*) The description of the authentication scheme.
Protection level	setLevel(*value*) A value of 1 through 1000. The higher the number, the greater degree of protection provided by the scheme. Default is 15.
Library	setLibrary("smauthcert") The default library for this scheme type.
Parameter	setParameter(*param*) A string containing the domain or IP address of the SSL server and the name and path of the SSL Credentials Collector (SCC). The server redirects a user's X.509 certificate over an SSL connection. Format: https://*server:port/SCC*?cert+basic The following example uses the default SCC: https://my.server.com:80/siteminderagent/cert/smgetcred.scc?cert+basic

Information Type	Value Assignment and Meaning
Shared secret	setSecret("") Set to an empty string. Not applicable to this scheme.
Is template?	setIsTemplate(0) Set to false (0) to indicate that the scheme is not a template.
Is used by administrator?	setIsUsedByAdmin(0) Set to false (0)—scheme is not used to authenticate administrators.
Save credentials?	setAllowSaveCreds(0) Set to false (0) to indicate that user credentials won't be saved.
Is RADIUS?	setIsRadius(0) Set to false (0)—scheme is not used with RADIUS agents.
Ignore password check?	setIgnorePwCheck(*flag*) Set to true (1) to ignore password checking, or false (0) to check passwords. Default is 0.

X.509 Client Cert and Form Template

Use this table when configuring an authentication scheme based on the scheme type X.509 Client Certificate *and* Form. The Java methods referenced in the table are in the class SmScheme.

Information Type	Value Assignment and Meaning
Scheme type	setType(TypeX509ClientCertAndForm) The scheme type X.509 Client Certificate and HTML Form.
Description	setDescription(*description*) The description of the authentication scheme.
Protection level	setLevel(*value*) A value of 1 through 1000. The higher the number, the greater degree of protection provided by the scheme. Default is 15.
Library	setLibrary("smauthcert") The default library for this scheme type.

Information Type	Value Assignment and Meaning
Parameter	setParameter(*param*) A string containing the domain or IP address of the SSL server and the name and path of the forms credentials collector (FCC). The server redirects a user's X.509 certificate over an SSL connection. Format: https://*server:port*/*FCC*?cert+forms The following example uses the default FCC: https://my.server.com:80/siteminderagent/certoptional/forms/login.fcc?cert+forms
Shared secret	setSecret("") Set to an empty string. Not applicable to this scheme.
Is template?	setIsTemplate(0) Set to 0 to indicate that the scheme is not a template, or 1 if the scheme is a template. Default is 0.
Is used by administrator?	setIsUsedByAdmin(0) Set to 0—scheme is not used to authenticate administrators.
Save credentials?	setAllowSaveCreds(0) Set to 0 to indicate that user credentials won't be saved.
Is RADIUS?	setIsRadius(0) Set to 0—scheme is not used with RADIUS agents.
Ignore password check?	setIgnorePwCheck(*flag*) Set to 1 to ignore password checking, or 0 to check passwords. Default is 0.

X.509 Client Cert or Basic Template

Use this table when configuring an authentication scheme based on the scheme type X.509 Client Certificate *or* Basic. The Java methods referenced in the table are in the class SmScheme.

Information Type	Value Assignment and Meaning
Scheme type	setType(TypeX509ClientCertOrBasic) The scheme type X.509 Client Certificate or Basic.
Description	setDescription(*description*) The description of the authentication scheme.

Information Type	Value Assignment and Meaning
Protection level	setLevel(*value*) A value of 1 through 1000. The higher the number, the greater degree of protection provided by the scheme. Default is 5.
Library	setLibrary("smauthcert") The default library for this scheme type.
Parameter	setParameter(*param*) A string containing the following information: Server for establishing an SSL connection. This server redirects a user's X.509 certificate over an SSL connection. Name and path of the SSL Credentials Collector (SSC). If you are using basic authentication over SSL, also provide the following two pieces of information: The fully qualified name of the SSL server used for establishing an SSL connection for basic authentication. Name and path of the SSL Credentials Collector (SSC). https://*SSLserver:port*/*SCC*?certorbasic; [https://*BasicServer*/*SCC*]
	The following example uses the default SCC values: https://my.SSLserver.com:80/siteminderagent/certoptional/smgetcred.scc?certorbasic; https://my.BasicServer.com/siteminderagent/nocert/smgetcred.scc
Shared secret	setSecret("") Set to an empty string. Not applicable to this scheme.
Is template?	setIsTemplate(0) Set to false (0) to indicate that the scheme is not a template.
Is used by administrator?	setIsUsedByAdmin(0) Set to false (0)—scheme is not used to authenticate administrators.
Save credentials?	setAllowSaveCreds(0) Set to false (0) to indicate that user credentials won't be saved.

Information Type	Value Assignment and Meaning
Is RADIUS?	setIsRadius(0) Set to false (0)—scheme is not used with RADIUS agents.
Ignore password check?	setIgnorePwCheck(*flag*) Set to true (1) to ignore password checking, or false (0) to check passwords. Default is 0.

X.509 Client Cert or Form Template

Use this table when configuring an authentication scheme based on the scheme type X.509 Client Certificate *or* Form. The Java methods referenced in the table are in the class SmScheme.

Information Type	Value Assignment and Meaning
Scheme type	setType(TypeX509ClientCertOrForm) The scheme type X.509 Client Certificate or HTML Form.
Description	setDescription(*description*) The description of the authentication scheme.
Protection level	setLevel(*value*) A value of 1 through 1000. The higher the number, the greater degree of protection provided by the scheme. Default is 5.
Library	setLibrary("smauthcertorform") The default library for this scheme type.
Parameter	setParameter(*param*) A string containing the following information: ■ Server for establishing an SSL connection. This server redirects a user's X.509 certificate over an SSL connection. ■ Name and path of the SSL and forms credentials collector (SFCC). If you are using an alternate forms-based authentication over SSL, also provide the following two pieces of information: ■ The fully qualified name of the SSL server used for establishing an SSL connection for authentication. ■ Name and path of the Forms Credentials Collector (FCC). https://*SSLserver:port/SFCC*?certorform; [https://*BasicServer/FCC*]

Information Type	Value Assignment and Meaning
	The following example uses the default SCC values: https://my.SSLserver.com:80/siteminderagent/certoptional/forms/login.sfcc?certorform; https://my.BasicServer.com/siteminderagent/forms/login.fcc
Shared secret	setSecret("") Set to an empty string. Not applicable to this scheme.
Is template?	setIsTemplate(0) Set to 0 to indicate that the scheme is not a template, or 1 if the scheme is a template. Default is 0.
Is used by administrator?	setIsUsedByAdmin(0) Set to 0—scheme is not used to authenticate administrators.
Save credentials?	setAllowSaveCreds(0) Set to 0 to indicate that user credentials won't be saved.
Is RADIUS?	setIsRadius(0) Set to 0—scheme is not used with RADIUS agents.
Ignore password check?	setIgnorePwCheck(*flag*) Set to 1 to ignore password checking, or 0 to check passwords. Default is 0.

X.509 Client Cert Template

Use this table when configuring an authentication scheme based on the scheme type X.509 Client Certificate. The Java methods referenced in the table are in the class SmScheme.

Information Type	Value Assignment and Meaning
Scheme type	setType(TypeX509ClientCert) The scheme type X.509 Client Certificate.
Description	setDescription(*description*) The description of the authentication scheme.
Protection level	setLevel(*value*) A value of 1 through 1000. The higher the number, the greater degree of protection provided by the scheme. Default is 5.

Information Type	Value Assignment and Meaning
Library	setLibrary("smauthcert") The default library for this scheme type.
Parameter	setParameter(*param*) A string containing the domain or IP address of the server responsible for establishing the SSL connection and the name and path of the SSL Credentials Collector (SCC). The server redirects a user's X.509 certificate over an SSL connection. Format: https://*server*/*SCC*?cert The following example uses the default SCC value: https://my.server.com/siteminderagent/cert/smgetcred.scc?cert
Shared secret	setSecret("") Set to an empty string. Not applicable to this scheme.
Is template?	setIsTemplate(0) Set to false (0) to indicate that the scheme is not a template.
Is used by administrator?	setIsUsedByAdmin(0) Set to false (0)—scheme is not used to authenticate administrators.
Save credentials?	setAllowSaveCreds(0) Set to false (0) to indicate that user credentials won't be saved.
Is RADIUS?	setIsRadius(0) Set to false (0)—scheme is not used with RADIUS agents.
Ignore password check?	setIgnorePwCheck(1) Set to true (1)—ignore password checking.

Performance Consideration

The following properties of the SmRealm object are set to true by default:

- *PropProcessAuthEvents*. When true, authentication event processing occurs.
- *PropProcessAzEvents*. When true, authorization event processing occurs.

Authentication and authorization event processing affect performance. If no rules in the realm are triggered by authentication or authorization events, set the associated property to false.

Chapter 5: Authentication and Authorization APIs

This section contains the following topics:

Configuration of All Custom Classes

The following configuration information applies to all custom authentication schemes and active expressions implemented with the Java Authentication API and Java Authorization API:

- The library name is always smjavaapi.
- In the parameter field, the first item must be the name of the custom class you implemented with the Authentication API or Authorization API, as follows:
  - With authentication schemes, specify the name of the class you implemented from the base interface SmAuthScheme. The class name should include the fully qualified package name, such as:

    com.myorg.sdk.myclass
  - With active policies, active rules, and active responses, specify the name of the class you implemented from the base interface ActiveExpression.
- If any parameters are specified in the parameter field after the class name, the class name is separated from the parameters list by a space character.

  When SiteMinder calls the methods in an instance of your custom class, it passes the specified parameters. The class name is not passed. The parameters are passed as a single string. If the string contains multiple parameters, the parameters can be delimited in any way that the custom class requires.
- The class file specified in the parameter field must be referenced in the classpath directive of the JVMOptions.txt configuration file. This file is located in Netegrity/SiteMinder/Config within the SiteMinder installation path.
- With active expressions, the function name (that is, the entry point for the smjavaapi library file) is always JavaActiveExpression.

Custom Classes for Authentication and Authorization

The basic steps for implementing and deploying custom authentication or authorization classes are as follows:

1. Implement the custom authentication or authorization class using the Authentication or Authorization API and the common classes.
2. Deploy the custom class or jar file on the Policy Server machine, and specify its location in the classpath directive of the JVMOptions.txt file. This file is located in Netegrity/siteminder/config within the SiteMinder installation path.
3. Configure the custom authentication or authorization functionality in the Policy Server User Interface.

Required Library File

All custom authentication and authorization classes use the same library file—smjavaapi. This library file is included with the Policy Server. You do not have to modify this library file. You simply reference it when you are configuring your custom authentication or authorization class.

Shared Information

Custom authentication and authorization objects may sometimes need to communicate request-specific information between themselves, such as to preserve state between object instances. These objects can share information through AppSpecificContext, which is retrieved through ApiContext. ApiContext is one of the common classes that is passed to both authentication and authorization objects.

Information shared through AppSpecificContext has request-only scope. For example, a custom object running in the context of an authentication request cannot exchange information with an object running in the context of an authorization request.

Common Classes

The following classes are used by both the Authentication API and the Authorization API. The services that these classes provide include:

- Sending logging, tracing, and error messages to the Policy Server
- Providing a mechanism for custom authentication and authorization objects to share information
- Making user context information available to authentication and authorization objects

The following table summarizes the common classes:

Class	Description
APIContext	Allows logging, tracing, and error messages to be sent to the Policy Server.
AppSpecificContext	Provides methods that allow custom authentication and authorization objects to share information.
SmJavaApiException	Provides exception functionality to custom authentication and authorization objects.
UserContext	Allows a custom object to set and retrieve information about a user in a user directory. The information includes user attributes and directory attributes associated with the user. The methods for setting and retrieving user directory attributes are available only if isUserContext() returns true.

Create a Custom Authentication Scheme

Authentication schemes provide a way to collect a user's credentials and determine the user's identity.

The Policy Server includes a variety of standard authentication schemes. These schemes range from basic user name/password authentication and HTML forms-based authentication to digital certificate and token authentication.

If the standard authentication schemes included with the Policy Server do not provide the kind of authentication functionality required at your site, you can use the Java Authentication API to create a custom authentication scheme.

Classes and Interfaces in the Authentication API

The base interface in the Java Authentication API is SmAuthScheme. All custom authentication schemes created with the Java Authentication API must implement this interface.

SmAuthScheme Methods

SiteMinder calls the following methods in the base interface SmAuthScheme:

Method	Description
authenticate()	Performs the custom authentication and returns the authentication result. SiteMinder calls this method at least twice—to establish user context and to authenticate the user's credentials.
init()	Performs any initialization procedures that the authentication scheme requires. SiteMinder calls this method once for each authentication scheme instance, when the authentication scheme is loaded.
query()	Provides SiteMinder with either of the following kinds of information, depending on the value SiteMinder passes in the *reason* parameter (object SmAuthQueryCode): The version and description of the authentication scheme. The kind of credentials that SiteMinder should collect from the user, and optionally, the URL for the site where credentials should be collected.
release()	Performs any rundown procedures that the authentication scheme requires. SiteMinder calls this method once for each authentication scheme instance, when SiteMinder is shutting down.

Other Classes in the Authentication API

The following classes are used in conjunction with the SmAuthScheme base interface:

Class	Description
SmAuthenticationContext	Contains the following context classes passed to authenticate(): APIContext UserContext UserCredentialsContext The set... methods in this object pass information directly to SiteMinder.
SmAuthenticationResult	Contains status and reason codes returned to SiteMinder after a call to authenticate().

Class	Description
SmAuthQueryCode	Contains the type of request that SiteMinder is making of the authentication scheme (version number and description, or information about the credentials that SiteMinder must collect). SiteMinder passes this object to the authentication scheme in query().
SmAuthQueryResponse	Contains constants that specify the kind of credentials that are required for authentication, if any. Also allows a text message to be returned to SiteMinder for display to the user. Optionally, the authentication scheme can call setResponseBuffer() to specify a URL where credentials must be collected. The set... methods in this object pass information directly to SiteMinder.
SmAuthStatus	Contains Authentication API status codes such as SMAUTH_ACCEPT and SMAUTH_REJECT. This object can be passed back to SiteMinder in SmAuthenticationResult. It is also the return value type for the authentication scheme's init(), query(), and release() methods.
UserCredentialsContext	Contains credentials information and other information from the user directory where user context was established. This object is contained in the SmAuthenticationContext object passed to authenticate().

How SiteMinder Loads a Custom Authentication Scheme

An authentication scheme verifies the user credentials that SiteMinder passes to it and returns an authentication result.

When user authentication is to occur against a custom authentication scheme created with the Java API, SiteMinder loads the custom scheme by loading:

- The standard library file smjavaapi that is installed with the Policy Server
- An instance of the custom class implemented from SmAuthScheme

How SiteMinder Initializes Authentication Processing

Immediately after the scheme is loaded, SiteMinder calls the following methods in the custom class implemented from SmAuthScheme:

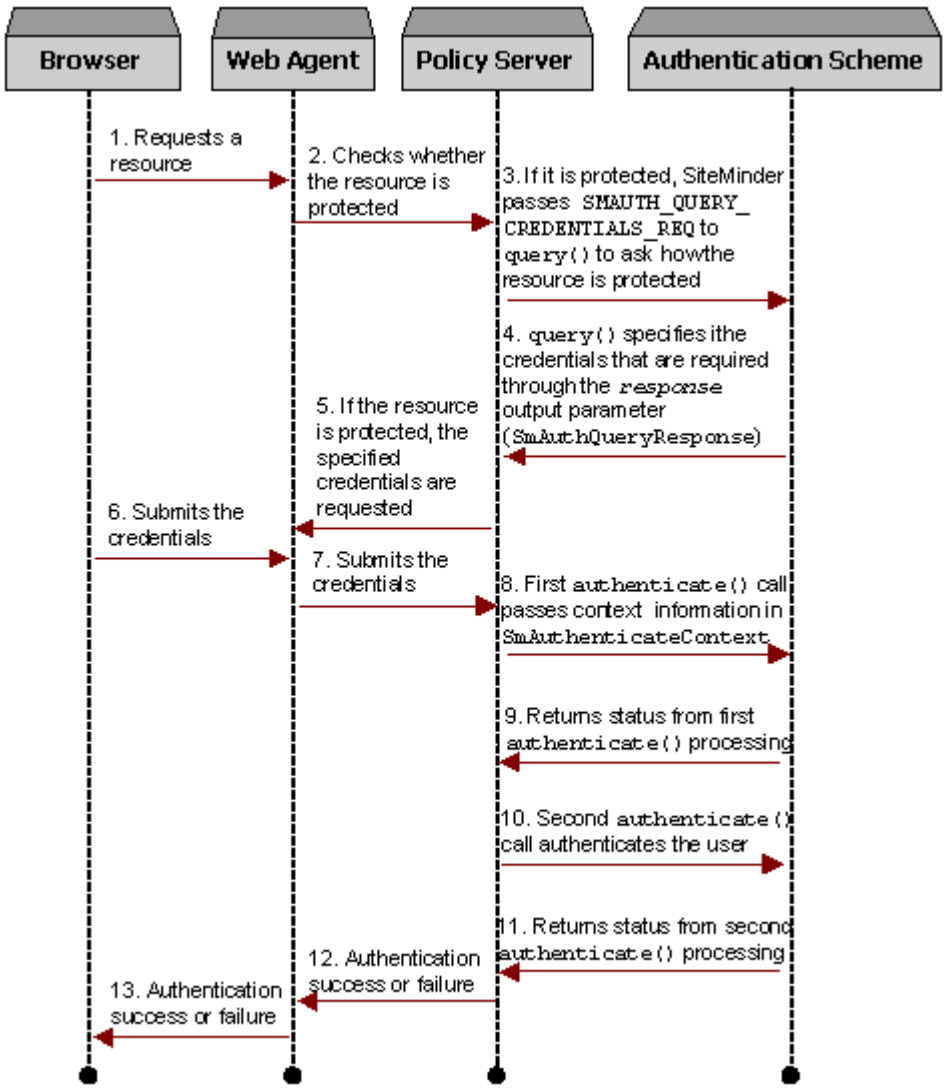
- query(). SiteMinder passes SMAUTH_QUERY_DESCRIPTION in the *request* parameter, requesting that the authentication scheme pass back the version number and description of the Java Authentication API.

  Note: When SiteMinder passes SMAUTH_QUERY_CREDENTIALS_REQ in query(), SiteMinder is requesting that the authentication scheme specify the kind of credentials that are required. SiteMinder then collects the specified credentials.

- init(). SiteMinder passes the parameter string and shared secret that were defined when the authentication scheme was configured in the Policy Server. The scheme then performs any initialization procedures it requires.

Authentication of User Credentials

The following figure shows the key activities that occur during authentication:



Supported Credentials

The Java Authentication API supports authentication based on the following general types of credentials requirements:

- Username/Password
- X.509 Certificate
- Custom user attributes

You specify the authentication credentials that are required through the setResponseBuffer() method in the object SmAuthQueryResponse. This object contains a number of constants that indicate the specific credentials that are required or whether no credentials are required.

User Disambiguation and Authentication

The authentication process includes two phases—*user disambiguation* and *user authentication*.

Before a user can be authenticated, the user's profile information must be retrieved from the user store so that the user's stored credentials can be compared with the credentials supplied at login. Looking up the user in a user store (such as an LDAP user directory or an ODBC database) is called *user disambiguation*. Either SiteMinder or the authentication scheme can disambiguate the user.

SiteMinder calls SmAuthScheme.authenticate() at least once during the disambiguation phase and at least once during the authentication phase:

- During disambiguation, it is called once per directory where disambiguation occurred.
- During authentication, it is called once per user found in the directory.

The basic steps are as follows:

1. **User login**. The user supplies a login ID (such as jsmith) for authentication purposes.
2. **Disambiguation phase**. Before the user lookup in the data store can begin, a complete DN or a search expression must be constructed based upon the supplied login ID. For example, if the login ID is jsmith, the DN used to search the user store might be constructed as follows:

   `uid=`**`jsmith`**`,ou=marketing,o=myorg.org`

   An LDAP search expression can also be used to search an LDAP user directory, and a SQL query is used to search an ODBC database—for example:

   `(&(objectclass=inetOrgPerson)(uid=`**`jsmith`**`))`

   select Name from SmUser where Name = '**jsmith**'

   Multiple results are possible, given that the LDAP DN or the ID stored in the ODBC database might apply to different users who have different passwords.
3. **Authentication phase**. The custom authentication scheme compares the known credentials of each disambiguated user with the credentials supplied during login—for example, by comparing the hash of the supplied password against the hash in the user store.

User Disambiguation

SiteMinder first calls authenticate() at the beginning of the user disambiguation phase.

Either SiteMinder or the custom authentication scheme can disambiguate the user. The authentication scheme indicates whether it has performed the disambiguation through a combination of the following:

- One of the status codes listed below
- Whether a value is passed to SiteMinder in the setUserText() method of SmAuthenticationContext

The status codes are set in the SmAuthStatus object. This object is passed in the *status* parameter of the SmAuthenticationResult constructor. SmAuthenticationResult is returned from authenticate():

- SMAUTH_NO_USER_CONTEXT

  The authentication scheme asks SiteMinder to disambiguate the user.

  When returning this status code, the authentication scheme should also return an empty string through the setUserText() method. SiteMinder gets the login ID from the Agent, constructs the DN or search expression based on the login ID and the information defined in the SiteMinder User Directory Properties dialog box, and disambiguates the user by looking up the user in the user store.

- SMAUTH_SUCCESS

  The authentication scheme asks SiteMinder to disambiguate the user.

  The authentication scheme passes the login ID to SiteMinder through setUserText(). SiteMinder uses that value to construct the DN or search expression and disambiguate the user in the user store. This approach gives the authentication scheme the opportunity to modify the login ID before SiteMinder disambiguates the user.

  Note: If the authentication scheme passes an empty string in setUserText(), SiteMinder uses the login ID provided by the Agent (the same behavior as with return code SMAUTH_NO_USER_CONTEXT).

- SMAUTH_SUCCESS_USER_DN

  The authentication scheme disambiguates the user by constructing the complete DN or search expression and looking up the user in the user store. The authentication scheme passes the user's complete DN or ODBC database ID to SiteMinder in setUserText(). Only one DN or database ID can be passed in setUserText().

- SMAUTH_ATTEMPT.

  The user cannot be found in the directory.

- SMAUTH_FAILURE

  This is returned if an error condition exists. Error text is returned to SiteMinder through the setUserText() method.

User Authentication

During this phase, SiteMinder calls authenticate() again to allow the authentication scheme to verify the supplied credentials after the user context has been established during disambiguation. The method sets one of the following status codes:

- SMAUTH_ACCEPT. The user is authenticated.
- SMAUTH_REJECT. The user is not authenticated.
- SMAUTH_CHALLENGE. The user is challenged. The scheme passes the challenge message to SiteMinder through the setUserText() method. Also, a reason code must be supplied in the SmAuthenticationResult object returned by authenticate().
- SMAUTH_FAILURE. An error condition occurred. Error text is passed to SiteMinder in setUserText().

Redirection

Your authentication scheme can have the Policy Server instruct the agent to perform a redirect. To build redirection capabilities into your authentication scheme:

- Specify the redirection URL in:

  `SmAuthenticationContext.setErrorText()`

- When creating an SmAuthenticationResult object to return from authenticate(), specify REASON_ERROR_MESSAGE_IS_REDIRECT in the *reason* parameter of the constructor.

Authentication Events

Authentication results are tied to SiteMinder events. If authentication events are enabled in the realm where the user is being authenticated, SiteMinder evaluates optional policies tied to OnAuthAccept**,** OnAuthReject**,** OnAuthAttempt**,** and OnAuthChallenge rules. You can configure these policies to return custom responses based on a user's identity, redirect the user to another location based on the result of the authentication, or update the user data in an external database.

Extend the SAML and WS-Federation Authentication Schemes

The SiteMinder SAML (1.x and 2.0) and WS-Federation authentication schemes process response messages. For business reasons, for example, you might want to add additional steps to further process a response. The Message Consumer Extension API defines an interface that enables you to elaborate on the SAML or WS-Federation response in two ways during the authentication process:

- To report detailed failure reasons during user disambiguation
- To customize user credential validation

The SiteMinder Java MessageConsumerPlugin API implements the Message Consumer Extension (MCE) interface. You can code to your own requirements and then integrate the custom plug-in into SiteMinder.

The MessageConsumerPlugin includes the following four methods:

Method	Description
init()	Performs any initialization procedures that the plug-in requires. SiteMinder calls this method once for each plug-in instance, when the plug-in is loaded.
release()	Performs any rundown procedures that the plug-in requires. SiteMinder calls this method once for each plug-in instance, when SiteMinder is shutting down.
postDisambiguateUser()	Provides processing to disambiguate a user when the authentication scheme is unable to do so, or to add data for new federation users to a user store. Note that this method receives the decrypted assertion. The decrypted assertion is added to the properties map passed to MCP under the key "_DecryptedAssertion".
postAuthenticateUser()	Provides any additional code to determine the final outcome of assertion processing, regardless of whether the policy server processing results in success or failure.

SiteMinder provides the following samples of the Message Consumer plug-in class:

- MessageConsumerPluginSample.java in <install-path>\sdk\samples\messageconsumerplugin
- MessageConsumerSAML20.java in <install-path>\sdk\samples\authextensionsaml20

The Role of the MessageConsumerPlugin

The following list describes the MessageConsumerPlugin in an elaborated process of user authentication:

1. The Federation Web Services (FWS) application forwards a request for user authentication to the Policy Server.
2. The Policy Server invokes the authentication scheme to disambiguate the user.
3. The authentication scheme disambiguates the user as follows:
    a. The authentication scheme metadata is obtained from the Policy store.
    b. The authentication scheme attempts to obtain the LoginID. If LoginID is not found, the authentication scheme invokes the MessageConsumerPlugin as described in Step 4.
    c. If the LoginID is obtained successfully, the authentication scheme searches the current user directory with a predefined SearchSpec. If the user is not found, the postDisambiguate() method is called as described in Step 4. If the user is found, the Policy Server proceeds with credential validation, as described in Step 6 and following.
    d. When the authentication scheme does not provide the user store SearchSpec, the Policy Server core searches for the user with the search string defined with the User Directory object. The MessageConsumerPlugin is not called.
4. The postDisambiguateUser() method searches a user directory to determine whether a particular LoginID exists. The result is returned to the authentication scheme. The method might be called several times if more than one user directory is configured. This method can also be used to add data for new federation users from the assertion to a user store.
5. When the user has been successfully disambiguated by the Policy Server, the authentication scheme, or the plug-in, the Policy Server returns the user DN to the Policy Server and proceeds to credential validation (Step 8 and following).
6. If the user has not been successfuly disambiguated for this user directory by either the Policy Server, the authentication scheme, or MessageConsumerPlugin, the FWS application checks the next user directory and repeats Steps 2 - 6 before proceeding with credential validation.
7. When a user has been disambiguated, the Policy Server again calls the authentication scheme to determine whether the user has the proper credentials for the authentication request. The authentication scheme determines whether the response message is acceptable.
8. After the the authentication scheme has attempted to authenticate the disambiguated user with the response message, the Policy Server calls the postAuthenticateUser() method from the MessageConsumerPlugin. The Policy Server always calls this method when a user is disambiguated, even when the Policy Server core performs the user disambiguation.

9. You can use postAuthenticateUser() to add any other procedures for federation credential validation required by your implementation.
10. The final result is passed back to the Policy Server by the authenticaiton scheme.
11. If necessary, the FWS application can process any failure and redirect the user to an appropriate URL.

Use the Authorization API

The Java Authorization API lets you implement custom functionality for controlling access to protected resources.

The functionality is provided through custom Java classes that are referenced in Policy Server active expressions. An *active expression* is a string of variable definitions that appears in the following Policy Server objects:

- Active policy—A policy that provides dynamic authorization based on external business logic.

  For example, you might implement a custom Java class that returns true if the user belongs to a particular organizational unit (ou) in an LDAP directory. The ou is passed to the custom Java class in the parameter (*param*) field of the active expression.

- Active response—A response returned from a custom Java class. Using an active response is one way you can define user-specific privilege information.

  For example, you might define an active response that returns a user's common name (cn) if the user belongs to the ou passed in the *param* field of the active expression.

- Active rule—A rule that provides dynamic authorization based on external business logic.

  For example, you might define a custom Java class that returns true if a user is a member of a group, such as Directory Administrator, that has permission to view a realm. The group name is passed to the Java class in the *param* field of the active expression.

Active Expressions

Active expressions are constructed in the Policy Server User Interface using the following syntax:

```
<@ lib=<lib-spec> func=<func-spec> param=<func-params>@>
```

An active expression based on the Java Authorization API has the following required fields:

- *lib* contains the shared library name smjavaapi. This library is used with all active expressions that reference a custom Java class in *param*.
- *func* contains the function name JavaActiveExpression. This function is the entry point for the smjavaapi library. It is used with all active expressions that reference a custom Java class in *param*.
- *param* contains the following information.
  - The name of your custom Java class
  - Optionally, any parameters to pass to an instance of your class

You define an active expression when you configure the active policy, rule, or response in the Policy Server User Interface.

Execute an Active Expression

When SiteMinder detects an active expression that references a custom Java class, it performs the following tasks:

- Loads the shared library and instance of the custom Java class specified in the active expression.
- Calls the library function specified in the active expression.
- Passes to the instance of the custom Java class the optional parameter string plus the following context objects:
  - APIContext
  - RequestContext
  - UserContext
- The instance of the Java class performs the custom functionality and returns a result to SiteMinder. Results are returned from the custom Java class's invoke() method.

Interpret an Active Expression Result

SiteMinder interprets the result returned by the instance of the custom Java class according to the type of active expression that references the Java class, as follows:

- Active Policy—If the result returned is an empty string or if an exception is thrown, authorization is denied.

  The policy does not fire if the result returned matches any of the following strings (not case-sensitive): FALSE, F, or 0.

  Any other result causes the policy to fire.

- Active Rule—If the result returned is an empty string or if an exception is thrown, the following behavior occurs:

  - With Allow Access rules, the rule does not fire.
  - With Deny Access rules, the rule fires.

  Otherwise, the behavior is the same as for Active Policies.

- Active Response—The result is a string that corresponds to a response attribute. How SiteMinder interprets the result string is determined by the response attribute specified in the Policy Server User Interface. For example:

  - WebAgent-OnReject-Redirect. Given this response attribute, SiteMinder expects the result string to specify a location, such as a URL, to redirect a user who is denied access to a resource.

    (The URL that is passed back might vary according to information passed into the custom Java class. For example, a group name could be passed in the *param* field of the active expression. The custom Java class could then test for the group name to determine the URL to pass back.)

  - WebAgent-HTTP-Cookie-Variable. Given this response attribute, SiteMinder expects that the result string, such as the user's common name, is to be assigned to a cookie variable. You can use the result string any way you like, such as to display the user's common name to personalize a form.

    You specify the cookie name in the SiteMinder Response Attribute Editor.

  If the method fails (that is, the method returns -1 or 0), the response attribute is ignored.

Classes and Interfaces in the Authorization API

The base interface in the Java Authorization API is ActiveExpression. All Java classes that provide custom authorization functionality must implement this interface.

The name of the class that you implement from the base interface must appear in the *param* field of any associated active expression.

ActiveExpression Methods

The base interface in the Java Authorization API is ActiveExpression. All Java classes that provide custom authorization functionality must implement this interface.

The name of the class that you implement from the base interface must appear in the *param* field of any associated active expression.

SiteMinder calls the following methods in the base interface ActiveExpression:

Method	Description
init()	Performs any initialization procedures that the custom Java class requires. SiteMinder calls this method once per instance of the custom ActiveExpression class.
invoke()	Performs the custom authorization functionality in the ActiveExpression object and returns a result.
release()	Performs any rundown procedures that the ActiveExpression object requires. SiteMinder calls this method once for each instance of an ActiveExpression class, when SiteMinder is shutting down.

Note: Classes that implement ActiveExpression should be implemented on a stateless model that does not depend on instance state stored in member variables of the ActiveExpression class.

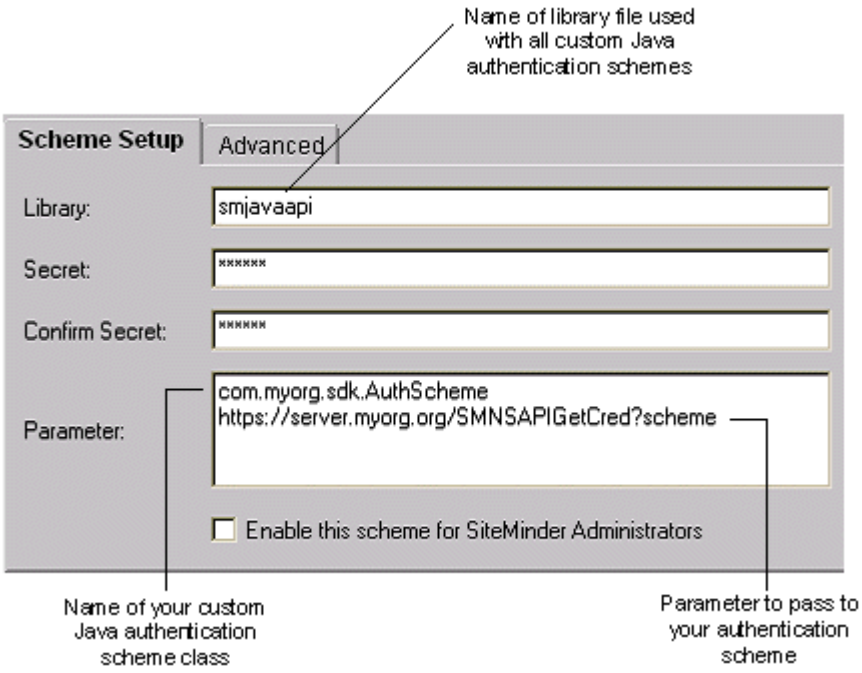
Other Classes in the Authorization API

The following classes in the Authorization API are used in conjunction with the ActiveExpression base interface:

Class	Description
ActiveExpressionContext	Contains the following context classes passed to invoke(): ■ APIContext ■ RequestContext ■ UserContext
RequestContext	Provides information about the user's access request—for example, the server or resource portion of the request.

Custom Authentication Scheme Configuration

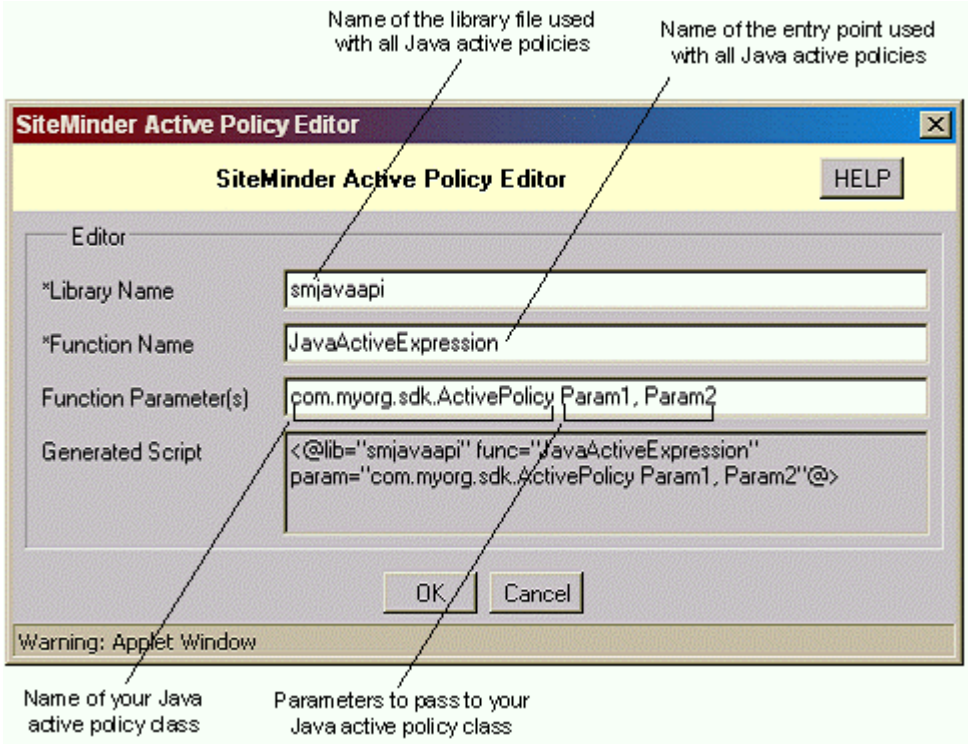
The following diagram is an example of an authentication scheme configuration defined on the SiteMinder Authentication Scheme Dialog. The Authentication Scheme Type is Custom Template:



To configure a custom authentication scheme on the SiteMinder Authentication Scheme Dialog, access the dialog box, click the Advanced tab, and select Custom Template as the Authentication Scheme Type.

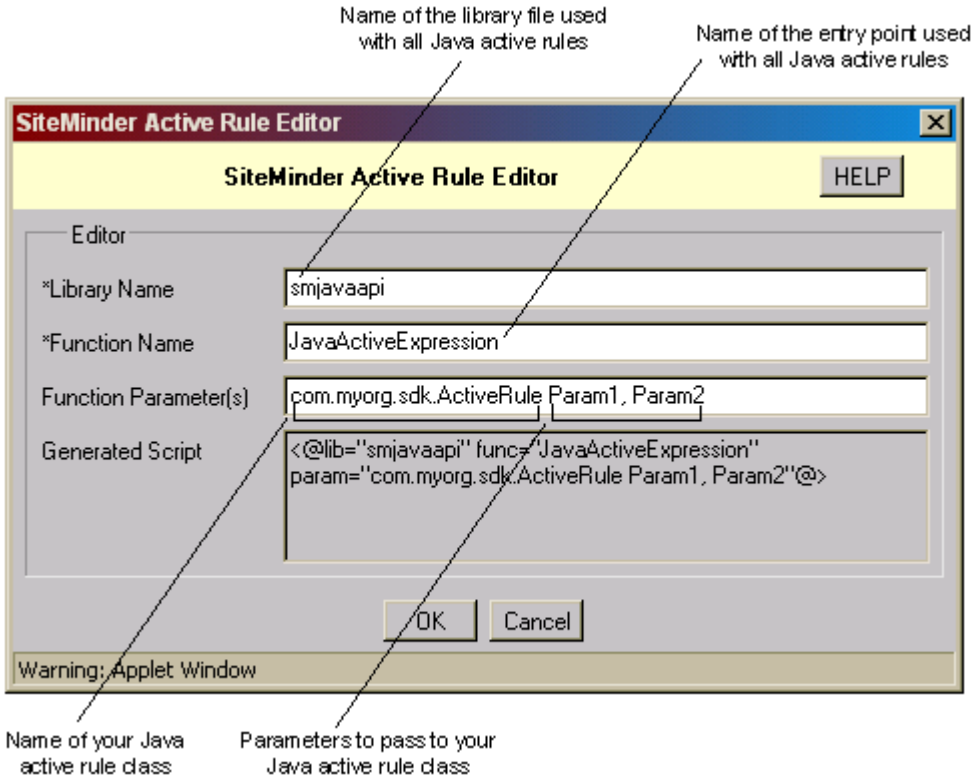
Active Policy Configuration

The following diagram is an example of an active policy configuration defined in the SiteMinder Active Policy Editor:



Active Rule Configuration

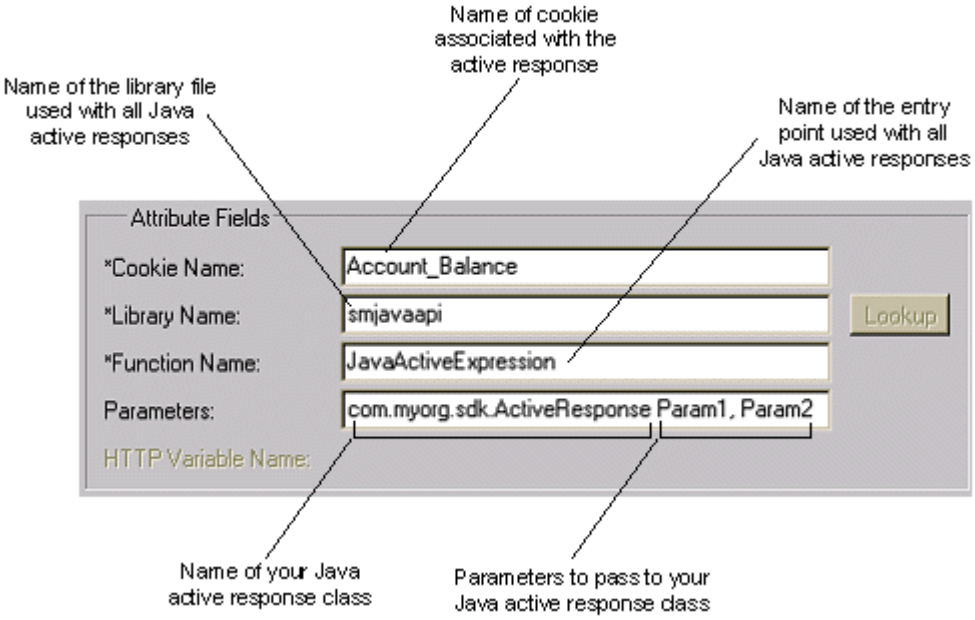
The following diagram is an example of an active rule configuration defined in the SiteMinder Active Rule Editor:



To access the editor from the Rule Properties dialog box, select the Active Rule tab in the Advanced group box, then click Edit.

Active Response Configuration

The following diagram is an example of an active response configuration defined in the SiteMinder Response Attribute Editor. In the example, the active response is associated with a WebAgent-HTTP-Cookie-Variable attribute:



From the Response Properties dialog box, access the editor by clicking Create. Then, in the Attribute Setup tab, select the Active Response button in the Attribute Kind group box.

Modify a SAML Assertion or Response

According to SAML specifications, an assertion (SAML 1.x) or response (SAML 2.0) is generated by a producer site and sent to a consumer site for validation. Typically, you will use the default SAML assertion or response that SiteMinder generates at the producer site. If you want to modify the content of the assertion or the reponse, you can do so by implementing the Java assertion generator plug-in. This plug-in is appropriate for both consumers (SAML 1.x) and Service Providers (SAML 2.0).

To modify the SAML assertion or response

1. Implement a Java SAML assertion generator plug-in.

   The implementation is a plug-in for the SiteMinder Assertion Generator Framework. The Assertion Generator Framework sends a default token to the custom plug-in object. After processing, the custom object passes a modified token to the Assertion Generator Framework.

2. Configure the plug-in by specifying the fully-qualified name of the plug-in class and any optional parameters that the plug-in might require.

   You configure a custom assertion generator plug-in in any of these ways:

   - For SAML 1.x support, on the Advanced tab of the SiteMinder Affiliate Properties dialog.
   - For SAML 2.0 support, on the Advanced tab of the SiteMinder Service Provider Properties dialog.
   - Through the C or Perl Policy Management API.

   Note: Configuration of the assertion generator plug-in requires a Policy Management API session version of at least v6.0 SP 2.

Interaction between SiteMinder and an Assertion Generator

The following steps outline the interaction between SiteMinder and a custom assertion generator plug-in. The activities begin when an authorized user makes a request, through a SiteMinder Policy Server, for a resource at a site that consumes assertions:

1. An authorized user requires a SAML assertion or response for a consumer or Service Provider.
2. The SiteMinder Assertion Generator Framework generates a default SAML assertion or response.
3. If an assertion generator plug-in is defined for the site that consumes the assertion, the SiteMinder Assertion Generator Framework requests an instance of the plug-in object from the plug-in cache.

   Note: The site consuming assertions can have no more than one assertion generator plug-in defined for it.

4. If the plug-in has not yet been loaded into cache:

   a. SiteMinder instantiates the plug-in class and loads it into cache.

   b. SiteMinder calls the plug-in's init() method. This method performs any initialization procedures that you have implemented for the plug-in.

   A successfully initialized plug-in object remains in cache until SiteMinder shuts down. This avoids having to re-load and re-initialize the object every time the plug-in is required.

5. The SiteMinder Assertion Generator Framework passes the default XML token, generated in step 2, to the plug-in's customizeAssertion() method.

6. The plug-in validates or modifies the information as required, and returns the processed assertion to the Assertion Generator Framework.
7. The Assertion Generator Framework passes the processed token to the consumer or Service Provider. This site uses the information in the assertion to determine how to respond to the user's request.
8. Steps through are repeated whenever a user requires an assertion for the service provider.

When SiteMinder is about to shut down, SiteMinder calls the plug-in's release() method to allow the plug-in to perform any rundown activities it might require.

Development and Deployment Notes

When developing and deploying a custom assertion generator plug-in, keep the following points in mind:

- The implemented plug-in class must provide a public default constructor method with no parameters.
- The implementation must be stateless—that is, the single plug-in instance can be concurrently used for multiple threads.
- The syntax requirements and use of the parameter string that is passed into the customizeAssertion() method is the responsibility of the custom object.
- The custom object must parse the default assertion that is passed to customizeAssertion()—for example, through a Document Object Module (DOM) parser or a Simple API for XML (SAX) parser. The sample plug-in class AssertionSample.java uses a DOM parser.
- To enable SiteMinder to locate your deployed plug-in class, specify the deployment location in -Djava.class.path of the configuration file JVMOptions.txt. This file is in the following location of the SiteMinder installed directory structure: <install-path>\siteminder\config.
- For a sample of an assertion plug-in class, see AssertionSample.java, which demonstrates the plug-in process for a SAML 1.x assertion, in the following location of the SiteMinder installed directory structure: <install-path>\sdk\samples\assertiongeneratorplugin. In the same directory there are samples for the Assertion Generator Plugin for SAML 2.0, SAML2AppAttrPlugin.java, and WS-Federation, WSFedAppAttrPlugin.java.

Chapter 6: Delegated Management Services API

This section contains the following topics:

About the DMS API

Directory management consists of managing objects within a SiteMinder user directory. For example, a user of your directory management application can create organizations, add groups to organizations, and add end users to groups. Your application performs directory management operations with the DMS API.

The Delegated Management Services (DMS) API lets you perform directory management operations on LDAP and ODBC directories.

With LDAP directories, you can use the DMS API to write a client application that allows a user with the specified privileges to perform tasks such as (but not limited to):

- Creating an organization
- Creating a group
- Adding a group to an organization

- Adding a user to a group
- Modifying the profile of a user

With ODBC directories, you can perform many but not all DMS API operations.

Note: The DMS API (available in Java only) has different functionality than the DMS Workflow API (available in C/C++ only). The DMS API lets you develop directory management applications that perform similar operations as the SiteMinder DMS product. The DMS Workflow API works in conjunction with DMS and fires when certain pre-process and post-process DMS events occur, allowing you to develop applications that perform additional functionality before and/or after these events.

The Required JAR File

The JAR file smjavasdk2.jar is required for building and running Delegated Management applications. The JAR file is stored in the following locations:

- Windows platforms:

  <install_path>\sdk\java

- UNIX platforms:

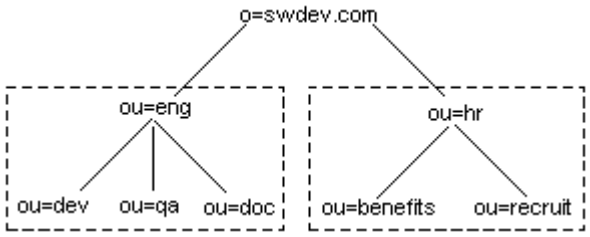
  <install_path>/sdk/java

SiteMinder User Directories

A SiteMinder user directory is a conceptual view of a single organizational unit (such as Engineering or Human Resources) within a larger entity (such as a corporation). SiteMinder user directories make managing an entire directory structure easier by breaking up the directory into smaller, more manageable, and logically related segments.

The methods in your custom DMS application reference a particular SiteMinder user directory by specifying its unique organization DN. The organization DN points to the root, or top level, of the SiteMinder user directory's inverted tree structure or to one of its sub-levels.

Every DMS request references an organization DN. In the following illustration, two SiteMinder user directories are enclosed in broken-line boxes. The directories are identified by the organization DNs ou=eng, o=swdev.com (representing the Engineering organizational unit) and ou=hr, o=swdev.com (representing the Human Resources organizational unit):



SiteMinder user directories can exist within other SiteMinder user directories. In the preceding illustration, the Engineering organizational unit has three SiteMinder user directories within it. These have the attribute and organization names ou=dev, ou=qa, and ou=doc. The Human Resources organizational unit has two SiteMinder user directories within it—ou=benefits and ou=recruit.

SiteMinder User Directory Containers

An organization DN in a SiteMinder user directory typically has one or more sub DNs. Sub DNs are also called "containers" because they contain lists of information. The default names of these containers and the information they contain are:

- people—End users in the Siteminder user directory.
- roles—Roles used in the SiteMinder user directory.
- groups—Groups used in the SiteMinder user directory.
- orgadmin—Group for administrators who can manage the organizational unit.

Sub DNs are managed by the class SmDmsConfig. When you create an SmDmsConfig object, you can keep the default sub DN names or assign new ones.

Organization administrators are listed in the orgadmin container. In a hierarchical organization, an organization administrator listed in a given orgadmin container can manage the organizational unit associated with that container and any organizational units below it.

Attribute-based Delegation

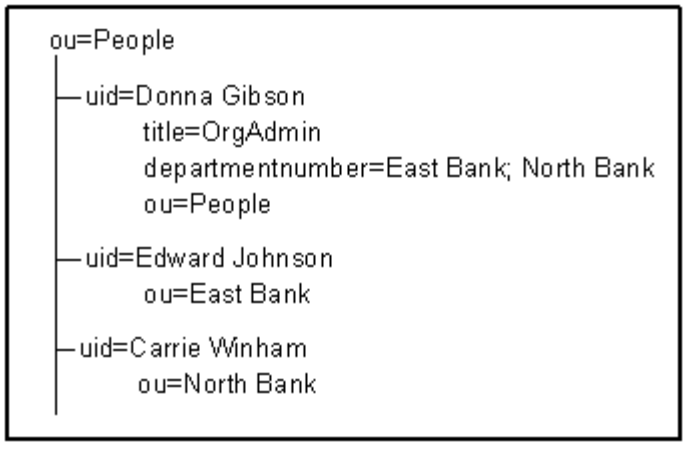
In addition to hierarchical organization, DMS also provides an administration model for sites that have implemented a flat directory structure. In this model, delegation is based on attributes in user profiles instead of hierarchical levels.

In a flat directory, DMS adds attribute/value pairs to user profiles to group users together. Once users are grouped together, another attribute/value pair determines which users can manage the groups.

DMS groups users into organizations by adding an attribute/value pair to user profiles. For example, users who belong to the organization East Bank have the attribute/value pair *ou*=East Bank in their profiles, where *ou* is the attribute that indicates the organization to which a user belongs.

An organization administrator can only manage organizations that are listed in the organization administrator's profile. The list of organizations is assigned to a profile attribute that you specify in the SmDmsConfig constructor. For example, if you specify departmentnumber as the attribute that contains the organizations that an organization administrator can manage, the attribute/value pair *departmentnumber*=East Bank means that the organization administrator can manage the East Bank organization and no others.

The following illustration describes how attribute-based delegation is implemented:



In this example, Donna Gibson is an organization administrator for East Bank and North Bank. She can manage Edward Johnson and Carrie Winham because they belong to organizations that are listed in the departmentnumber attribute in Donna's user profile.

Configure Attribute-based Delegation

You specify the attributes that enable attribute-based delegation in the SmDmsConfig constructor. Three attributes are required to identify the following information:

- A user as an organization administrator. This attribute can be any attribute in your LDAP directory that you are not using to store other information—for example, o.

  You specify a user as an organization administrator through the constructor's *OrgAdminSubDn* parameter, as in the following example:

  OrgAdminSubDn="(title=OrgAdmin)";

- The organization(s) that an organization administrator can manage. By default, DMS uses the departmentnumber attribute to store managed organizations.

  You specify this attribute through the constructor's *OrgAdminOrgs* parameter—for example:

  OrgAdminOrgs="departmentnumber";

- The organization to which a user belongs. By default, DMS uses the ou attribute.

  You specify this attribute through the constructor's *DnOrgs* parameter—for example:

  DnOrgs="ou";

DMS Users

DMS users are assigned one of the following categories of directory management privileges. The categories are listed below from lowest to highest:

- End user—Can manage certain information about the end user's own account, such as changing the user's password and viewing (but not adding) the roles that the user is a member of.
- Organization administrator—Can manage an entire organization and any organizations below it.
- SiteMinder administrator—Can manage directories in one or more domains.

  SiteMinder administrator privileges can vary. With DMS, SiteMinder administrators must have system-level Manage User privileges, and they must be present in at least one domain.
- Super administrator—Can manage all directories in all domains.

You use different login() methods to log in different categories of DMS users.

Implementation Class

Interface SmDmsApi is implemented by the class SmDmsApiImpl. Use this class as the starting point for the DMS API.

This class lets you determine how you want to access the information in the SmDmsDirectory object. You can do so by providing either of two kinds of information:

- The name or OID of the target user directory. To provide this information, call getDirectoryContext().
- The OID of the protected realm that the user is attempting to access. To provide this information, call getDmsContext().

These methods fill the context object that is passed into them.

Context Class

The getDirectoryContext() and getDmsContext() methods in class SmDmsApiImpl create a context object—either SmDmsDirectoryContext or SmDmsContext. The context object contains information such as user directory, session, and connection information. The context object is so-named because its information is derived within the context of the provided realm OID or the user directory name or OID. When you have a context object, you call its getDmsDirectory() method to retrieve an SmDmsDirectory object. This object represents an LDAP or other namespace and gives you access to organizations and other elements in the namespace.

Object Class

The Object class, SmDmsObject, and its subclasses provide methods for creating and managing directory objects. SmDmsObject includes the following subclasses:

- SmDmsDirectory represents a user namespace, such as LDAP. It provides access to the information in an entire directory.
- SmDmsOrganization represents an organization, such as Engineering or Human Resources, within a directory. A SiteMinder user directory is a conceptual view of an organization. It is managed by an organization administrator and uniquely identified by an organization DN.
- SmDmsGroup represents a group within an organization. Groups are sets of objects that have something in common—for example, a group of employees who have been with the company for less than a year. With group objects, users can be assigned privileges collectively instead of individually.
- SmDmsRole represents a role within an organization. A role describes a user's function in an organization. This allows the user to be managed with other users who have the same privileges. For example, a user who can order items online and view an inventory list may have the role buyer.
- SmDMSUser represents a user within an organization. Users can be end users or administrators.

Object Model

When performing an operation on a directory, organization, group, role, or user object, you sometimes have a choice of using the generic SmDmsObject or one of its subclasses. However, for object-specific operations (such as authenticating a user, changing a user's password, or getting a user's privileges), you have to use an object-specific subclass.

The objects corresponding to the subclasses are distinguished by a *class identifier*, such as DMSOBJECT_CLASS_USER for a user object. These identifiers are defined in SmDmsObject. When you create an object using a subclass, such as creating a user with SmDmsUser, and then you call addObject(), the class identifier is automatically set. However, if you create a generic directory, organization, group, role, or user object with SmDmsObject, you must set the class identifier before calling addObject().

Search Class

The Search class, SmDmsSearch, represents a configuration object for the search operation. It holds the search base and the filter. The filter expects a string-based search expression for the object class.

The search class returns a list of distinguished names paired with the corresponding class identifier, and optionally, selected attribute information for the items retrieved in the search.

Cursor Class

The SmDmsCursor class lets you define sorting and paging behavior for result set operations—for example:

- Set the *sort* parameter of the SmDmsCursor constructor to specify the columns to use for sorting rows.
- Call setBlockSize() to define the maximum number of rows that can be returned from a result set at one time—that is, the maximum number of rows in a page.
- Call setOffset() to specify the starting offset (row number) of the block returned from the result set.
- Call isSortingCritical() to specify whether a result set must be sorted.
  - If you specify true, a result set will be retrieved only if it can be sorted.
  - If you specify false, an unsorted result set will be sent if it cannot be sorted. An isSorted() call on the same SmDmsCursor object will return false.
- Call isPagingCritical() to specify whether a result set must be paginated.
  - If you specify true, a result set will be retrieved only if it can be paginated.
  - If you specify false, a complete result set will be sent if it cannot be sorted. An isPaginated() call on the same SmDmsCursor object will return false.

Searches that Support Cursor Operations

You can perform sorting and paging operations by passing a defined SmDmsCursor object into any of the following methods:

- One of the search methods in SmDmsOrganization:
  - search()
  - searchForward()
  - searchBack()
  - searchRefresh()
- SmDmsObject.getGroups()
- SmDmsOrganization.getGroups()
- SmDmsGroups.getMembers()

Note: getGroups() and getMembers() are not supported in searches of ODBC directories.

Searches of Microsoft LDAP Directories

Sorting and paging operations are not supported for Active Directories through the AD namespace. Sorting and paging operations are supported for Active Directories through the LDAP namespace.

When communicating with an Active Directory through the AD namespace, SiteMinder responds to sorting and paging requests as follows:

- If both isPagingCritical() and isSortingCritical() return false in the SmDmsCursor object, the result set is returned. No sorting and paging operations are performed.
- If either isPagingCritical() or isSortingCritical() returns true in the SmDmsCursor object, an error occurs. No result set is returned.

You specify whether sorting and paging operations are critical in the SmDmsCursor constructor.

Write a Directory Management Application

To write a Directory Management application

1. Establish a Connection to the Policy Server
2. Obtain a Session Object

   A session object is obtained when a user or administrator successfully logs in:

   - To log in a SiteMinder administrator and establish a SiteMinder administrator session, call the login() method in the SmApiSession class of the Utilities package.

     If login is successful, the session object contains the session specification.

   - To log in an end user, DMS organization administrator, or DMS super administrator, call the login() method in the AgentAPI class of the Agent API package.

     If the login is successful, the session specification is put into the *spec* field of the SessionDef object. Set the *spec* value in the SmApiSession object.

3. Pass in the Session Object

   After obtaining a valid session, create a DMS API object by passing the session to the constructor of the SmDmsApiImpl class—for example:

   ```
   SmDmsApi dmsApi = new SmDmsApiImpl (apiSession);
   ```

   In the example, dmsApi is the new DMS API object, and *apiSession* is the session obtained when the administrator successfully logged in.

   Note: Whenever you create a DMS API object, you pass the session and connection information to the object.

4. Create a Directory Management Context

   To use the DMS API to access a user directory, you need to know either:

   - The OID of a realm that has a self-registration scheme configured for it.

     Call SmDmsApiImpl.getDmsContext() to pass in this information.

   - The SiteMinder user directory where you are operating.

     Call SmDmsApiImpl.getDirectoryContext() to pass in this information.

   The type of information you know or choose to provide determines the directory management context for accessing the user directory, as follows:

If You Know...	And...	Then Use...
The OID of a realm that contains a self-registration scheme	The user is a SiteMinder administrator	Delegated Management Services (DMS) context

If You Know...	And...	Then Use...
The SiteMinder user directory name or OID	—	Directory context

DMS context and directory context provide two different avenues for reaching the same destination—an SmDmsDirectory object where you can access and manipulate directory information.

5. Create and Manipulate Objects

   After creating a context, you can create and manipulate directory objects using the DMS Object Model. When working with directory objects, you need to know:

   - The distinguished name of the object.
   - The type of object, such as:
     - Top-level organization
     - Organizational unit
     - Group
     - User
     - Role

DMS Context

DMS context lets you access an SmDmsDirectory object within the context of a realm OID that you provide. The DMS context class is SmDmsContext.

You can create a DMS context object as follows:

```
SmDmsContext dmsContext = new SmDmsContext();
```

You can retrieve a DMS context object, use the method getDmsContext() in the class SmDmsApiImpl.

Note: SiteMinder administrator privileges are required for calling getDmsContext().

Before retrieving the DMS context object information, you need to create a realm object to pass into the getDmsContext() call. The realm object must:

- Have a valid object identifier (OID) obtained from an agent call to AgentAPI.isProtected().
- Be configured with a registration scheme.

You create the SmRealm object as follows:

```
SmRealm realm = new SmRealm();
```

Then, set the realm OID by calling setOid(). You can call this method through an object that extends the SmObjectImpl class of the Policy Management API.

After setting the OID for the realm object, call getDmsContext() and pass in the realm object.

Example:

An agent calls isProtected() to determine if the resource that a user is attempting to access is protected. The Policy Server indicates that the resource is protected by returning the credentials required for accessing the resource. Included with the return information is the OID of the protected realm. As shown in the example below, you use the returned realm OID (in the example, m_REALM_OID) to set the OID for the realm object you are creating and passing to getDmsContext():

```
// Create a DMS API object from a valid session.
SmDmsApi dmsApi = new SmDmsApiImpl (apiSession);

// The realm below should contain a registration scheme.
// You can get a directory OID from the registration scheme.
SmRealm realm = new SmRealm ();
realm.setOid (m_REALM_OID);
// Create the DMS context object.
SmDmsContext dmsContext = new SmDmsContext ();

// This call returns the realm, self registration,
// and user directory information through dmsContext.
result = dmsApi.getDmsContext (realm,
                               new SmDmsConfig(),
                               dmsContext);
```

To get complete directory information from the dmsContext object, call dmsContext.getDmsDirectory().

To get just the directory OID, call dmsContext.getSelfReg(), and then call SmSelfReg.getUserDir().

Directory Context

Directory context lets you access an SmDmsDirectory object within the context of a user directory name or OID that you provide. The directory context class is SmDmsDirectoryContext. To get a directory context, use the method getDirectoryContext() in the class SmDmsApiImpl.

In the following example, an SmDmsDirectoryContext object is returned in *dirContext*. Call getDmsDirectory() to get the information about the directory object.

// Create a DMS API object from a valid session.

```
SmDmsApi dmsApi = new SmDmsApiImpl (apiSession);

// Create the directory context object.
SmDmsDirectoryContext dirContext=new SmDmsDirectoryContext();

// Directory object to pass in to getDirectoryContext().
SmUserDirectory userDir = new SmUserDirectory ();

// setOid() method can take the name of the user directory.
userDir.setOid ("smdev");

// This call returns directory information through dirContext.
result=dmsApi.getDirectoryContext(userDir,
                                  new SmDmsConfig(),
                                  dirContext);
```

Change the User Type in DMS Context

In a directory context, you can perform operations on behalf of any user type—super administrator, SiteMinder administrator, organization administrator, or end user. But to create a DMS context object, you must call the method getDmsContext(), and SiteMinder administrator privileges are required to call this method.

After getDmsContext() is called and DMS context is established for the session, it's possible to change the user type for subsequent operations in the session. For example, after a SiteMinder administrator opens a session in DMS context, you might want an end user to modify his user profile later in the same session. To make the profile request on the end user's behalf rather than the SiteMinder administrator's, you need to change the user type.

To create a DMS context object, you call SmDmsApiImpl.getDmsContext(). When you do so, connection information and the SiteMinder administrator's session specification are included the DMS context object.

As a chain of subsequent objects is created in the session (for example, SmDmsDirectory/SmDmsOrganization/SmDmsUser), the connection and session information is passed from object to object. To change the user type for a given object, you replace the SiteMinder administrator's session specification for that object with the session specification for the new user type on whose behalf subsequent calls will be made. You can change the session specification at any object level.

To change the user type for an object created in DMS context

1. Create the object that will be the target of requests by the new user type.

   For example, to make requests against the new user object dmsUser in organization dmsOrg on behalf of an end user with the distinguished name USER_DN:

   ```
   SmDmsUser dmsUser = dmsOrg.newUser(USER_DN);
   ```

   In the example, the SiteMinder administrator session specification in the dmsOrg object is passed to the dmsUser object.

2. Get a session specification for the new user in either of these ways:

   - With standard SiteMinder agents, use the default HTTP header HTTP_SM_SERVERSESSIONSPEC.
   - With custom agents, use the Agent API to log in the new user.

3. Pass in the session specification for the new user and DMS object. For example, if *sessionSpec* is the session specification:

   ```
   dmsUser.getApiSession().setSessionSpec(sessionSpec);
   ```

More Information:

Context Class (see page 136)

Create an Object

To create an object, such as an organization object, a group object, a user object, or a role object:

1. Use the context to get a directory object by calling getDmsDirectory() on a DMS context or directory context. For example, using a DMS context:

   ```
   SmDmsDirectory dmsDir = dmsContext.getDmsDirectory();
   ```

2. Use the directory object to create an organization object by calling newOrganization() in class SmDmsDirectory. Pass in the distinguished name of the organization, such as o=swdev.com. For example:

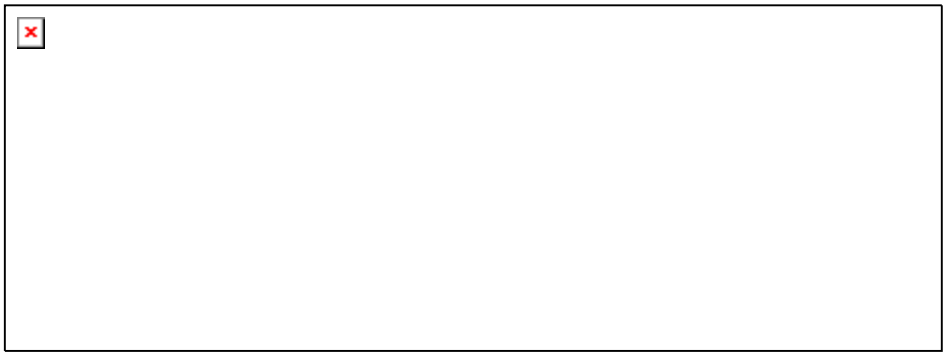
   ```
   SmDmsOrganization org=dmsDir.newOrganization("o=swdev.com");
   ```

3. Use the organization object to create other objects, such as group objects or organizational unit objects. The following example creates a group object named grp with the distinguished name ou=UI,ou=eng, o=swdev.com.

   ```
   SmDmsGroup grp=org.newGroup("ou=UI,ou=eng,o=swdev.com");
   ```

Note: This code does not add the group to the directory.

The following figure illustrates the DMS API flow for creating directory objects:



Get Directory Entry Attributes

To retrieve a value for a specific attribute, call getAttribute() in class SmDmsObject and pass in the attribute name as a string. Attribute values are available after you fetch the attributes with getObject(). The method getAttribute() returns a member of the java.lang.Object class. If the attribute is multi-valued, the returned object will contain multiple values delimited by a caret (^).

Add an Object to a Directory

To add an object to a directory:

1. Set the attributes for the object by calling setAttribute() in class SmDmsObject and passing to it the attribute name and its value. Attribute names are defined in your directory system.

   Call setAttribute() as many times as necessary to define the object.

2. Call the method addObject() in class SmDmsObject. For example:

   ```
   result = grp.addObject();
   ```

   In the example, *result* is an SmApiResult object.

   Note: If you want to call addObject() on a (generic) SmDmsObject object, you must first call setClassId() to set the class identifier.

When adding an object, you can set multiple values for the *objectclass* attribute, but not for other attributes. When modifying an object with the modifyObject() method, you can set multiple values for any attribute.

To set multiple values for an attribute, you can either:

- Pass in a string, using a caret (^) to delimit the values.
- Pass in a vector of values and have SiteMinder convert the vector to a string.

For example, to pass in a string containing the values top and organizationalunit, you could use the following code:

```
group.setAttribute("objectclass","top^organizationalunit");
```

To pass in a vector for the same values, you could use the following code:

```
Vector objectclass = new Vector();
objectclass.add("top");
objectclass.add("organizationalunit");
group.setAttribute("objectclass", objectclass);
```

Note: For existing objects, object class can be modified through the modifyObjectClass() method. This method also allows you to set multiple values for object class.

Add a User to a Group

To add a user to a group, call the addToGroup() method in class SmDmsObject. In the following example, the user *user1* is added to the group *devGroup*:

```
SmDmsDirectory dmsDir = dmsContext.getDmsDirectory();
SmDmsOrganization org = dmsDir.newOrganization(ORG_ROOT);
SmDmsGroup devGroup = org.newGroup(GROUP_DN);
SmDmsUser user1 = org.newUser(USER_DN1);
result = devGroup.addToGroup(user1);
```

Add a User to a Role

To add a user to a role, call the addToRole() method (class SmDmsUser). In the following example, the user *user1* is added to the role *role*:

```
SmDmsDirectory dmsDir = dmsContext.getDmsDirectory();
SmDmsOrganization org = dmsDir.newOrganization(ORG_ROOT);
SmDmsRole role = org.newRole(ROLE_DN);
SmDmsUser user1 = org.newUser(USER_DN1);
result = user1.addToRole(role);
```

Get, Modify, or Delete an Object

To get or modify an object's attributes, or to delete an object, call getObject(), modifyObject(), or deleteObject(). These methods are defined in class SmDmsObject.

For example, to get the attributes of the organization org whose DN is referenced by ORG_ROOT in the directory namespace dmsDir:

```
ORG_ROOT="o=swdev.com";
SmDmsDirectory dmsDir = dmsContext.getDmsDirectory();
SmDmsOrganization org = dmsDir.newOrganization(ORG_ROOT);
SmApiResult result = org.getObject();
```

To modify an object's attributes, you first fetch the existing attributes with getObject(). Then, you set the new attribute(s) by calling setAttribute() (in class SmDmsObject), just as you do when adding an object.). For example, to modify the user USER_DN1 in the organization org above by setting attribute *l* to the value Boston:

```
SmDmsUser user = org.newUser(USER_DN1);
result = user.getObject();
user.setAttribute("l", "Boston");
result = user.modifyObject();
```

You can modify multiple values for all attributes, not just the *objectclass* attribute.

To delete the user in the previous example:

```
SmDmsUser user = org.newUser(USER_DN1);
result = user.deleteObject();
```

Enhance DMS Performance

When using most of the DMSAPI calls, the Policy Server verifies that the user performing the operations is an administrator of the directory it is trying to modify, search, or update. The Policy Server does that by fetching all the domains and their associated user directories. The Policy Server verifies whether the user is the administrator of the directory. This process takes time. When there are multiple threads trying to verify administrators at the same time, the contention for the locks on the cache slows down the operation

The DMSAPI includes alternative methods that do not verify the administrator for each directory. The administrator is verified only once at login. The alternative methods also do not fetch all the domains and user directories.

The alternative methods are listed following. They are implemented by adding a parallel method with the 'NoAdminCheck' suffix. Use these methods when you are looking for enhanced performance. The original methods are still available when you require administrator validation.

Note: See the Javadoc for details about these methods.

SmDmsApi class

getDmsContext/getDmsContextNoAdminCheck

SmDmsDirectory class

getCapabilities/getCapabilitiesNoAdminCheck

getDmsRoles/getDmsRolesNoAdminCheck

getUserChallengeText/getUserChallengeTextNoAdminCheck

getUserTempPassword/getUserTempPasswordNoAdminCheck

getOrganizations/getOrganizationsNoAdminCheck

SmDmsOrganization class

getGroups/getGroupsNoAdminCheck

getRoles/getRolesNoAdminCheck

getOrgranizations/getOrganizationsNoAdminCheck

search/searchNoAdminCheck

searchForward/searchForwardNoAdminCheck

searchBack/searchBackNoAdminCheck

searchRefresh/searchRefreshNoAdminCheck

SmDmsUser class

changePassword/changePasswordNoAdminCheck

getDisabledState/getDisabledStateNoAdminCheck

setDisabledState/setDisabledStateNoAdminCheck

setEnable/setEnableNoAdminCheck

setDisable/setDisableNoAdminCheck

setPasswordMustChange/setPasswordMustChangeNoAdminCheck

getUserPWState/getUserPWStateNoAdminCheck

setUserPWState/setUserPWStateNoAdminCheck

authenticate/authenticateNoAdminCheck

getPrivileges/getPrivilegesNoAdminCheck

getRealmPrivileges/getRealmPrivilegesNoAdminCheck

getRealmContext/getRealmContextNoAdminCheck

addToRole/addToRoleNoAdminCheck

removeFromRole/removeFromRoleNoAdminCheck

getRoles/getRolesNoAdminCheck

getMembers/getMembersNoAdminCheck

SmDmsGroup class

getMembers/getMembersNoAdminCheck

SmDmsObject class

getObject/getObjectNoAdminCheck

modifyObject/modifyObjectNoAdminCheck

addObject/addObjectNoAdminCheck

deleteObject/deleteObjectNoAdminCheck

modifyObjectClass/modifyObjectClassNoAdminCheck

getGroups/getGroupsNoAdminCheck

addToGroup/addToGroupNoAdminCheck

removeFromGroup/removeFromGroupNoAdminCheck

Searches

You can search LDAP directories and ODBC directories. You search an organization using one of the search... methods in the class SmDmsOrganization.

You define a search using the following objects:

- SmDmsSearch to set search parameters such as the search starting point, the maximum number of records to retrieve, and the search filter. The following sections describe the use of this object.
- SmDmsCursor to define optional sorting and paging preferences.

You can specify the search parameters to use when searching the directory. There are two times when you can specify search parameters:

- When you create the search object
- After you create the search object

You can use either option or both options. They are not mutually exclusive.

Set Search Parameters When You Create the Search Object

To specify a search parameter when you create a search object, pass one or more search parameter names to the constructor of the SmDmsSearch class.

There are some search parameters that you cannot specify during creation of the search object—for example, scope. The constructor for the SmDmsSearch class accepts only the following search parameters:

- filter
- root
- propertyNames
- maxItems

You can create an SmDmsSearch object without passing any search parameters to the constructor.

Set Search Parameters After Creating the Search Object

After a search object is created, you can use the set... methods in the SmDmsSearch class to:

- Set additional search parameters.
- Reset search parameters that you set when the search object was created.

By using the set... methods, you can set or reset any of the parameters shown in the following table:

Parameter	Default	Set Method	Definition
classId	Unknown (not set yet)	setClassId()	Class identifier.
filter	" "	setFilter()	Search filter, or the string you want to find. Can also be set when the search object is created.
maxItems	50	setMaxItems()	Maximum number of result set items to display at a time. Can also be set when the search object is created.
nMaxResults	-1	setMaxResults()	Maximum number of items for the result set. For example, if nMaxResults is 500, but 750 items match the search criteria, only the first 500 matches will be returned from the search.
nextItem	-1	setNextItem()	The item to start with on the next search forward—for example: nextItem += maxItems
previousItem	-1	setPreviousItem()	The item to start with on the next search backward—for example: previousItem-=maxItems

Parameter	Default	Set Method	Definition
propertyNames	null	setPropertyNames( )	Properties to return from the search. Can also be set when the search object is created.
root	" "	setRoot()	Directory entry where the search should start. Can also be set when the search object is created. Valid for LDAP searches only.
scope	None	setScope()	Levels searched. For LDAP searches only.
timeout	-1	setTimeout()	Maximum duration of the search, in seconds.

Set the Search Filter

The search filter defines the items you want to retrieve in the search. You can set the search filter through an SmDmsSearch constructor or through the SmDmsSearch method setFilter().

The search filter is described differently for LDAP directories and ODBC directories.

Set the Search Filter for LDAP Directories

With LDAP directories, you provide a complete LDAP search filter in the *filter* parameter of an SmDmsSearch constructor or setFilter() method. For example, if you pass *filter* and *root* to the SmDmsSearch constructor to search the organization swdev.com for groups, you could specify the following:

```
SmDmsSearch search = new SmDmsSearch (
              "(&(objectclass=organizationalUnit) (ou=groups))",
               "o=swdev.com");
```

Set the Search Filter for ODBC Directories

A search of an ODBC directory is performed through a SQL query. The DMS API supports the SQL SELECT statement.

The information you provide in the search filter depends on whether your search uses an SmDmsCursor object to provide sorting and paging operations:

- With ODBC searches that do not pass an SmDmsCursor object to the search method, use a fully defined SQL SELECT statement in the search filter.
- With ODBC searches that do pass an SmDmsCursor object to the search method, use a partial SQL SELECT statement in the search filter, consisting only of FROM and WHERE clauses.

With ODBC database searches that pass an SmDmsCursor object to the search method, the DMS API constructs the complete SQL SELECT statement from various sources, as follows:

- The FROM and WHERE clauses are taken from the *filter* parameter of an SmDmsSearch constructor or setFilter() method.
- The SELECT *columns* portion of the query is taken from attributes specified in either of the following parameters:
  - The *propertyNames* parameter of setPropertyNames(). These attributes are used when an SmDmsSearch object is passed to one of the search methods in SmDmsOrganization.
  - The *attrNames* parameter of a getGroups() or getMembers() method.
- The ORDER BY *keywords* portion is taken from the order of the attributes you specify in the *sort* parameter of the SmDmsCursor constructor.

Consider the following code fragment:

```
String DIR_ROOT = "root";
String SRCH_FILTER ="from SmGroup";
SmDmsSearch search = new SmDmsSearch(SRCH_FILTER);
String[] prop = {"Name", "'Group' as Class"};
search.setPropertyNames(prop);
Vector SortOrder = new Vector();
SortOrder.add("uid");
SmDmsCursor cursor = new SmDmsCursor(SortOrder,blockSize,false,true);
```

The DMS API uses the information in the previous example to build the following SQL statement:

```
SELECT Name, 'Group' AS Class FROM SmGroup ORDER BY uid ASC
```

Code Source	**Portion of SQL Statement**
SRCH_FILTER parameter of SmDmsSearch constructor	from SmGroup
SortOrder parameter of SmDmsCursor constructor	order by uid asc
prop parameter of setPropertyNames()	select Name, 'Group' as Class

Search an Organization

In the DMS API, searches are performed on an organization object.

To search an organization:

1. Create a search object. This search object holds the search parameters.

   For example, the following SmDmsSearch constructor call creates a search object to search for groups. The *root* parameter specifies a start point of o=swdev.org.

   ```
   SmDmsSearch mySearch = new SmDmsSearch (
              "(&(objectclass=organizationalUnit) (ou=groups))",
               "o=swdev.org");
   ```

   Note: The root is the top level of the SiteMinder user directory to search. It is not necessarily the top level of the entire directory structure.

   Use the set... methods in the SmDmsSearch class to set any other search parameters—for example:

   ```
   mySearch.setScope(2);
   ```

2. Optionally, define sorting and paging preferences in the SmDmsCursor object.

3. Call the search() method in class SmDmsOrganization on the organization you want to search—for example:

   ```
   result = targetOrg.search (mySearch, 1);
   ```

   The second parameter of the search() method indicates the direction to search, as shown in the following table:

Direction	Integer Value
Reset	0
Forward	1
Back	2
Refresh	3

4. To get the items returned from the search, call getResults() on the search object—for example:

   ```
   Vector mySearchResults = search.getResults();
   ```

   The first element of the results vector contains the search parameters in a SmDmsSearchResultParams object. The remaining elements are SmDmsObject objects. To distinguish object types, the *classId* attribute of each object is set through the setClassId() method. For example, if the *classId* is DMSOBJECT_CLASS_USER, the object is a user. If the classId is DMSOBJECT_CLASS_GROUP, the object is a group.

Examples of a Search

The following example searches an organization using the search parameters set through the search.set... methods below. The results of the forward search are assigned to the vector *vsearch* and are printed along with the search parameters.

```
SmDmsContext dmsContext = new SmDmsContext();
SmDmsDirectory dmsDir = dmsContext.getDmsDirectory();
SmApiResult result = new SmApiResult();
SmDmsOrganization org = dmsDir.newOrganization (DIR_ROOT);

// Search
SmDmsOrganization test = org.newOrganization("");
SmDmsSearch search = new SmDmsSearch (
             "(&(objectclass=organizationalUnit) (ou=groups))",
              "o=swdev.com");
// Define search parameters
search.setScope(2);           // Number of levels to search.
search.setNextItem(0);        // Initialize forward search start
search.setMaxItems(20);       // Max number of items to display
search.setPreviousItem(0);    // Initialize back search start
search.setMaxResults(500);    // Max items in the result set
result = test.search(search, 1);
Vector vsearch = search.getResults();
System.out.println("Search object vector size " + vsearch.size());
SmDmsSearchResultParams searchParams =
                 (SmDmsSearchResultParams)vsearch.firstElement();
System.out.println("***Search Parameters***");
System.out.println(searchParams.toString());
System.out.println("removed element at 0");
vsearch.removeElementAt(0);
System.out.println("Search object vector size " + vsearch.size());
for (int i=0; i<vsearch.size(); i++)
{
   SmDmsObject dmsObj = (SmDmsObject)vsearch.elementAt(i);
   System.out.println("***Search**** " + dmsObj);
   printObject (dmsObj, result);
}

Hashtable attrs = dmsObj.getAttributes();
Enumeration keys = attrs.keys();
Enumeration values = attrs.elements();
while(values.hasMoreElements() )
```

The following code fragment configures sorting and paging features through an SmDmsCursor object and performs a search. The parameters for the SmDmsSearch object search would be defined in the same way as in the previous example:

```
Vector SortOrder = new Vector();
SortOrder.add("uid");
int blockSize = 20;
SmDmsCursor cursor=new SmDmsCursor(SortOrder,blockSize,false,true);
cursor.setOffset(15);
result = org.search(search, cursor, 1);     //Forward search
System.out.println(keys.nextElement() + " = " +
                                          values.nextElement() );
```

User Password State

Password state refers to activities relating to a given user's password—for example, the last time the password was changed, and the last time the password was used to log in the user. To retrieve an existing SmDmsUserPWState object for a user, or to set a new password state object with any attribute changes, call getUserPWState() or setUserPWState() in SmDmsUser.

The following table lists the password state attributes you can access for a given user, and the method used to set or retrieve an attribute value. All methods are in the class SmDmsUserPWState, unless otherwise noted.

Password State Attribute	Method	Description
Login failures	setLoginFailures() getLoginFailures()	Sets or retrieves the number of times the user failed to log in since the user's last successful login.
Last login time	setLastLoginTime() getLastLoginTime()	Sets or retrieves the time the user last logged in successfully.
Previous login time	setPrevLoginTime() getPrevLoginTime()	Sets or retrieves the next-to-last time the user logged in successfully.
Disabled time	setDisabledTime() getDisabledTime()	Sets or retrieves the time the user object was disabled.

Password State Attribute	Method	Description
Password history	SmDmsUser. setUserPWState()	Optionally, clears the user's password history when setting the password state object for the user. You cannot retrieve password history or set password history entries.
Last password change time	setLastPWChangeTime() getLastPWChangeTime()	Sets or retrieves the time the user's password was last changed.

If you change a password state attribute, the change applies to the current password state object only. To apply the change to a password state object that may be subsequently retrieved, pass the current password state object in a call to SmDmsUser.setUserPWState(). This method sets a new password state object containing the attribute values passed into the method.

ODBC Support

When operating against ODBC-based user directories, you can use the following DMS API methods:

- SmDmsApiImpl.getDirectoryContext(SmUserDirectory, SmDmsConfig, SmDmsDirectoryContext)
- SmDmsDirectory.getCapabilities(Vector)
- SmDmsDirectory.getUserChallengeText(String)
- SmDmsDirectory.getUserTempPassword(String, String)
- SmDmsObject.getGroups(Vector, boolean)
- SmDmsObject.getObject()
- SmDmsObject.getObject(Vector)
- All search... methods in SmDmsOrganization
- SmDmsUser.authenticate(String)
- SmDmsUser.changePassword(String, String, boolean)

DMS roles are not supported. Also not supported are operations such as adding and deleting users and groups, adding users to a group, and removing users from a group.

Restricted Methods

Some of the methods in the DMS API can only be called within a session established at a minimum level of the user privilege hierarchy or higher. For example, adding an end user to a role requires an organization administrator session, Siteminder administrator session, or super administrator session.

The following table shows the DMS methods (plus the login() and logout() methods in the apiutil package) that have security restrictions, the minimum privilege level required to call the methods, and the classes that the methods are called from:

Method	Minimum Privilege Level and Class
addObject()	Organization administrator session SmDmsObject and subclasses
addToGroup()	Organization administrator session SmDmsObject and subclasses
addToRole()	Organization administrator session SmDmsUser class
authenticate()	End user session SmDmsUser class
changePassword()	End user session SmDmsUser class
deleteObject()	Organization administrator session SmDmsObject and subclasses
getCapabilities()	End user session SmDmsDirectory class
getDirectoryContext()	End user session SmDmsApiImpl class
getDisabledState()	End user session SmDmsUser class
getDmsContext()	SiteMinder administrator session SmDmsApiImpl class
getDmsRoles()	Organization administrator session SmDmsDirectory class
getGroups()	End user session SmDmsObject and subclasses
getGroups()	Organization administrator session SmDmsOrganization class
getMembers()	Organization administrator session SmDmsGroup class

Method	Minimum Privilege Level and Class
getMembers()	Organization administrator session SmDmsRole class
getObject()	End user session SmDmsObject and subclasses
getOrganizations()	Organization administrator session SmDmsOrganization class
getRoles()	End User session SmDmsUser class
getRoles()	Organization administrator session SmDmsOrganization class
getUserChallengeText()	Super administrator session SmDmsDirectory class
getUserPWState()	End user session SmDmsUser class
getUserTempPassword()	Super administrator session SmDmsDirectory class
login()	No session SmApiSession class
logout()	SiteMinder administrator session SmApiSession class
modifyObject()	End user session SmDmsObject and subclasses
removeFromGroup()	Organization administrator session SmDmsObject and subclasses
search()	Organization administrator session SmDmsOrganization class
searchBack()	Organization administrator session SmDmsOrganization class
searchForward()	Organization administrator session SmDmsOrganization class
searchRefresh()	Organization administrator session SmDmsOrganization class
setDisable()	Organization administrator session SmDmsUser class
setDisabledState()	Organization administrator session SmDmsUser class

Method	Minimum Privilege Level and Class
setEnable()	Organization administrator session SmDmsUser class
modifyObjectClass()	Organization administrator session SmDmsObject and subclasses
setPasswordMustChange()	End user session SmDmsUser class
setUserPWState()	End user session SmDmsUser class

Index

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W

X