CA™ SOA Security Manager

Programming Guide

r12.1

CA Product References

This document references the following CA products:

- CA™ SOA Security Manager
- CA™ SiteMinder® Web Access Manager

Contact CA

Contact Technical Support

For your convenience, CA provides one site where you can access the information you need for your Home Office, Small Business, and Enterprise CA 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 product documentation, you can send a message to techpubs@ca.com.

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

Contents

Chapter 1: SOA Security Manager SDK Overview

This section contains the following topics:

Introduction

The SOA Security Manager SDK provides two APIs:

Web Service Client API

A Java API that greatly simplifies the task of creating Web service consumer applications.

XML Agent Content Helper API

A Java API that lets you to create custom SOA Agents for Web Servers.

Support for Custom Code

CA supports the SOA Security Manager Software Development Kit (SDK) as part of our standard offerings. However, we do not support custom code written by customers or partners.

Customers who use the SDK must assume responsibility for the code they write. Valid support customers may ask brief "how-to" questions on a particular API. But if you require more in-depth assistance, such as design or architecture assistance, please contact CA Technology Services or CA Education to gain the knowledge or assistance you need.

CA Software Prerequisites

No SOA Security Manager or CA SiteMinder processes need to be running on the machine where you build custom applications using the SOA Security Manager APIs.

Further, no SOA Security Manager or CA SiteMinder software must be installed on the machine where you run custom applications built with the SOA Security Manager Java APIs.

Additionally, the SOA Security Manager Policy Server is required for running all custom XML-enabled agent applications (created using the Java XML Agent Content Helper APIs). The XML-enabled agent application runtime files can either be local or remote to the Policy Server.

Web Service Client API

To create a Web Service client application, a developer needs to utilize several technologies such as Simple Object Access Protocol (SOAP), HTTP(S), and XML D-Sig. Many of these technologies are still emerging and, being delivered by different vendors, can be very difficult to integrate.

The SOA Security Manager Web Service Client API brings all of these technologies together under a consistent and simple Java API, providing the following services to Web Service client application developers:

- **SOAP Support:**
  - Wrap a raw XML document in a SOAP envelope
  - Insert a SOAP Header
  - Create a SOAP action
- **HTTP(S) Support:**
  - Post XML document and set custom HTTP headers, receiving HTTP response, etc.
  - Support SSL, including passing of server and client certificates
- **XML-DSig Support**
- **Digitally sign SOAP and raw XML documents**
- **Validate XML digital signatures**
- **Certificate Support:**
  - Generate Certificate Signing Requests (CSRs)
  - Generate private and public keys
  - Create self-signed X.509 certificates

More information:

Using the Web Service Client API (see page 15)

XML Agent Content Helper API

The XML Content Helper API is a Java API that allows you to build custom XML-enabled agents that can authenticate and authorize XML documents posted to a processing application (such as a Web service) bound to a URL.

The XML Agent Content Helper API supplements the CA SiteMinder Java Agent API (part of the CA SiteMinder SDK). A custom agent that is built using these two APIs can protect any URL-bound application (such as a Web service), performing the following functions on XML messages posted to that application:

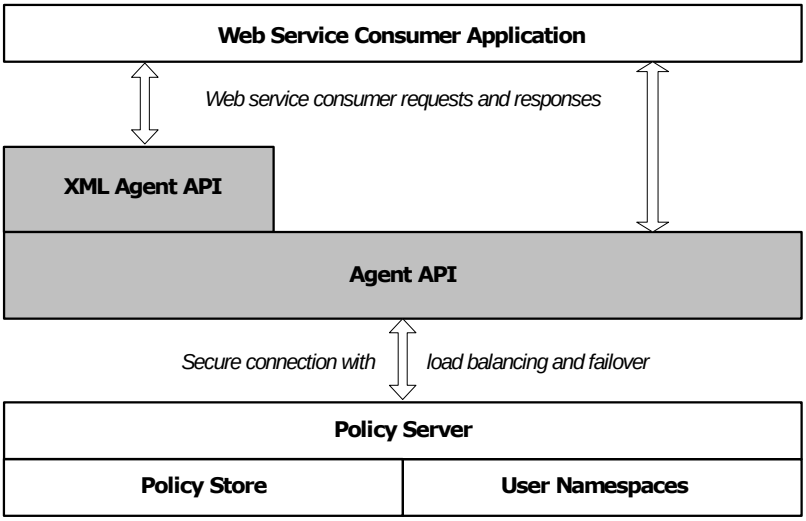
- Extracting credentials for authentication and authorization decisions
- Extracting body and header content for use in authorization decisions
- Adding SAML assertion responses

To build custom XML-enabled agents you need to intersperse functions from the two APIs to provide the necessary functionality.

Note: For more information about the services provided by the CA SiteMinder Agent API, refer to the *CA SiteMinder Web Access Manager Programming Guide for Java*.

Applications that are built using the CA SiteMinder Agent API and XML Agent Content Helper API are insulated from having to know specific implementation details about user accounts, privileges, and how to extract these from incoming XML messages. Instead, the two APIs work in combination with the Policy Server to greatly simplify application development while increasing application scalability with respect to the number of applications and resource-privilege pairs.

The following illustration shows the functional architecture of an XML-enabled Agent.



Further, the XML Agent Content Helper API insulates application developers from underlying XML message-based Web service technology details, including:

- Identifying requests for Web service resources
- Obtaining user credentials from the content of XML messages
- Resolving XML message content-based variables
- Adding SAML assertion responses to the content of XML messages and HTTP headers

What Is an XML-Enabled Agent?

An XML-enabled Agent is a client of the CA SiteMinder Agent API and XML Agent Content Helper API. XML-enabled Agents enforce SOA Security Manager XML message content-based access control policies served by the SOA Security Manager 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 Agents. Agents establish resource semantics and act as gatekeepers to protect resources from unauthorized users.

Different CA SiteMinder agent types protect different kinds of resources. Some agent types are pre-defined, standard agents that are shipped as part of the CA SiteMinder product—for example, the Web Agent, which provides HTTP access control for Web Servers. The SOA Agent for Web Servers, which ships as part of the SOA Security Manager product, is an XML-enabled version of the Web Agent that provides HTTP access control for Web services bound to URLs on a Web Server. However, you can also use the CA SiteMinder Agent API and SOA Security Manager XML Agent Content Helper API to implement custom agents.

When used with the CA SiteMinder Agent API, the XML Content Helper Agent API lets you create a custom XML-enabled agent that can authenticate and authorize XML messages posted to any URL-bound application in a variety of context-specific ways. For example, you could create an agent to protect an Application Server or a Message Queuing server.

XML Agent Content Helper API Services

The XML Agent Content Helper API provides agents with a set of services that supplement those provided by the CA SiteMinder Agent API to allow the development of sophisticated, secure, and robust XML-enabled agents. Building an XML-enabled agent involves using these services:

- Web service resource identification
- XML message credential extraction
- XML message variable resolution
- XML message response application
- XML message header and body retrieval
- XML message header and body content change notification

You will also require the following services provided by the CA SiteMinder Agent API:

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

Note: For more information on the services provided by the CA SiteMinder Agent API, see the *SiteMinder Web Access Manager Programming Guide for Java*.

SOA Security Manager SDK Contents

The Java API JAR file and sample source code are located under *SDK_Install_Dir*, the directory in which the SOA Security Manager SDK software is installed.

For example:

- C:\Program Files\CA\SOA Security Manager\sdk (Windows)
- ~/CA/SOA Security Manager/sdk (UNIX)

Note: For information about installing the SOA Security Manager SDK, see the *CA SOA Security Manager Implementation Guide*.

Installed Directory Structure

The SDK installation includes header files, binary files, library files, and examples, as shown in the following table.

Directory	Subdirectory	Files
/include	-	ContentHelperService.h
/bin	/linux	txmapi.so
	/solaris	txmapi.so
/samples	/javaClientAPI	XMLDocAgent.java xmldocagent.bat …
	/javaAgentAPI	HTTPApplication.java sampleapp.bat sampleapp.sh …
/lib	/win32	txmapi.lib
/java	-	soasmapi.jar

SDK Samples

The CA SOA Security Manager SDK contains a number of sample applications that can help you understand the APIs.

The samples are installed in subdirectories of the following directory:

SDK_Install_Dir/samples

The sample subdirectories contain source files, project files, makefiles, and other related files for building the sample applications and plug-ins.

The following table lists the subdirectories where the sample files are installed:

Subdirectory Name	Sample Description
javaAgentAPI	Java custom XML-enabled agent sample files.
javaClientAPI	Java Web service client application sample files.

SOA Security Manager SDK API Reference Material

The SOA Security Manager SDK includes a complete set of reference material for its APIs.

Javadoc Reference

For reference details such as syntax, parameter, return value, and exception information for the SOA Security Manager Java APIs, see the Javadoc.

You can access the Javadoc for all Java APIs in the SDK by opening the file index.html in the following default location:

Platform	File Locations
Windows	*SOA_HOME*\Documentation\ javadoc-tm
UNIX	*SOA_HOME*/Documentation/ javadoc-tm

Note: For information about installing the SOA Security Manager documentation, see the *CA SOA Security Manager Implementation Guide*.

Javadoc Version Information

The description of each package, class, and interface in the Javadoc includes a Since heading that indicates the SOA Security Manager 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.

Chapter 2: Using the Web Service Client API

This section contains the following topics:

Web Service Client API Members

This section provides a brief summary of the classes in the Java XML Agent API. Reference information on all the classes and methods, can be found in the Javadoc.

The Web Service Client API contains four classes that are contained in the package com.netegrity.tm.client.api.

The following table lists all four classes in the Web Service Client API.

Class/Interface	Description
XMLDocument	Wrapper class that encapsulates all XML document processing methods.
Utils	Wrapper class that encapsulates all utilities that assist in key and certificate generation.
HTTPClient	Class that provides methods to post XML documents over HTTP and obtain response message returned from a successful HTTP post.
HTTPSClient	Class that provides methods to post XML documents over HTTPS and obtain response message returned from a successful HTTPS post.

How to Implement an Application Using the Web Service Client API

Note: For a list of operating systems, Java environments, and platforms that the Web Service Client API supports, see the SOA Security Manager Platform Matrix on the Technical Support site at http://ca.com/support.

To implement an application the Web Service Client 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 the XML Agent Content Helper API JAR file is available whenever you compile or run a Web service client application built using the Web Service Client API. The JAR file, soasmapi.jar, is stored in the following locations:
   - Windows platforms:

     SDK_Install_Dir\Java
   - UNIX platforms:

     SDK_Install_Dir\Java

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

   For an example, see java-build.bat or java-build.sh in the sample directory smjavaagentapi.
6. Run the Web service client application.

More information:

SDK Samples (see page 12)

Sample Web Service Client API Java Application

SOA Security Manager provides a Sample Web Server Client Application that illustrates usage of the Web Service Client APIs. The Web Server Client Application is provided (along with its source code) to give you a better understanding of the APIs. It can also be used during development to test SOA Security Manager policies and web service, by allowing you to post test XML documents to your SOA Agent-protected web servers.

Set Up and Start the Sample Web Service Client Application

The following sections describes how to set up and run the Sample Web Service Client Application.

Perform Initial Setup

Before you run the Sample Web Service Client Application for the first time:

- (Windows) Edit the *SDK_install_dir*\samples\javaClientAPI\sampleapp.bat file:
  - Uncomment and modify the NETE_TXM_ROOT entry to specify the path to *SDK_Install_Dir* (for example, set NETE_TXM_ROOT=C:\Program Files\CA\SOA Security Manager\SDK).
  - Uncomment and modify the jre entry to specify the path to the jre subfolder of your JDK installation (for example, set jre=c\/jdk1.4.1\jre)
- (UNIX) Modify the *SDK_install_dir*/samples/javaClientAPI/sampleapp.sh file as follows:
  - Set the NETE_TXM_ROOT variable to point to *SDK_Install_Dir*
  - Set the java variable to point to the Java executable (java.exe), located in the bin subdirectory of your JDK installation.

Configure Values to be Used in the Sample Application

Optionally, you can specify values to be used to fill fields in the sample application by editing the txmclient.properties file. This can be particularly useful when you are using the sample application for testing purposes.

The txmclient.properties file can be found in:

- (Windows) *SDK_install_dir*\java
- (UNIX) *SDK_install_dir*/java

Run the Sample Application

To run the sample application

- (Windows) Run sampleapp.bat from *SDK_install_dir*\samples\javaClientAPI.
- (UNIX) Execute the sampleapp.sh script from *SDK_install_dir*/samples/javaClientAPI.

Source Code Location

The sample application source code can be found in the following locations:

- (Windows) *SDK_install_dir*\samples\javaClientAPI
- (UNIX) *SDK_install_dir*/samples/javaClientAPI

Chapter 3: Use the Java XML Agent Content Helper API

This section contains the following topics:

Overview

The XML Agent Content Helper API for Java supplements the CA SiteMinder Agent API for Java. A custom agent that is built using these two APIs can protect any URL-bound application (such as a Web service), performing the following functions on XML messages posted to that application:

- Extract credentials for authentication and authorization decisions
- Extract body and header content for use in authorization decisions
- Add SAML assertion responses

To build custom XML-enabled agents you need to intersperse functions from the two APIs to provide the necessary functionality.

Interfaces and Classes in the Java XML Agent Content Helper API

This section provides brief information about the classes in the Java XML Agent API. Reference information on all the classes and methods can be found in the Javadoc.

The primary point of access to the Java Agent API is the IContentHelper interface, implemented by ContentHelperService and contained in the package com.netegrity.tm.contenthelper.api.

The following table lists all the classes and interfaces in the XML Agent Content Helper API.

Interface/Class	Description
IContentHelper	Provides the primary methods for XML-enabling custom agents
IContentHelperReturnCodes	Provides possible return codes for methods in implementing XML Agent Content Helper API classes.
ContentHelperService	The boot strap for the XML Agent Content Helper API system. Use it to generate instances of the IContentHelper interface.
ServerVariablesResolver	Provides methods for setting and resolving SOA Agent variables (which provide information about the web server whose resources the XML-enabled agent is protecting) during authorization.
AgentKeys	Container class for CA SiteMinder agent keys (returned by the CA SiteMinder Agent API doManagement() method).
AgentApiService	Provides the XML Agent Content Helper API's connection to the Policy Server.

Implement the Java XML Agent Content Helper API

Note: For a list of operating systems, Java environments, and platforms that the XML Agent Content Helper API for Java supports, see the SOA Security Manager Platform Matrix on the Technical Support site at http://ca.com/support.

Code an XML-Enabled Java Agent

To write the source code for a custom XML-enabled agent using the XML Agent Content Helper API for Java

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.

More information:

SDK Samples (see page 12)
Java XML-Enabled Agent Functional Flow (see page 23)

Before You Compile an XML-Enabled Agent

Before you compile your agent application, ensure the following:

- 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\win32
  - **On Solaris:** Change LD_LIBRARY_PATH to include the following, so that libsmjavaagentapi.so can be found:

    install_path/sdk/bin/solaris
  - **On AIX:** Change LIBPATH to include the following, so that libsmjavaagentapi.so can be found:

    install_path/sdk/bin/aix
  - **On Linux:** Change LD_LIBRARY_PATH to include the following, so that libsmjavaagentapi.so can be found:

    install_path/sdk/bin/Linux

    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.so can be found:

    install_path/sdk/bin/hpux/hpux11

    Note: The Java Agent API is not available for HP10.
- The CA SiteMinder Agent API JAR file (smjavaagentapi.jar) is available by adding it to your CLASSPATH setting. When compiling, you can use the -classpath switch. smjavaagentapi.jar, is stored in the following locations:
  - Windows platforms:

    install_path\sdk\java
  - UNIX platforms:

    install_path/sdk/java

- The XML Agent Content Helper API JAR file is available whenever you compile or run an agent that uses the Java Agent API. The JAR file, soasmapi.jar, is stored in the following locations:
  - Windows platforms:

    Agent_install_location\Java
  - UNIX platforms:

    Agent_install_location\Java

  Add soasmapi.jar to your CLASSPATH setting. When compiling, you can use the -classpath switch.

Compile an XML-Enabled Java Agent

Compile the custom XML-enabled agent application using javac.

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

Deploy an XML-Enabled Java Agent

To deploy a custom XML-enabled Java Agent:

1. Do one of the following:
   - Install a SOA Agent for Web Servers on systems where the custom XML-enabled agent will run.
   - Copy the following files from a system on which a SOA Agent for Web Servers *is* installed into the same location on the system where the custom Agent will run:

     On Windows

     Agent_inst_dir\Java\soasmapi.jar

     Agent_inst_dir\Java\smjavaagentapi.jar

     Agent_inst_dir\Java\XmlSdkConfig.properties

     Agent_inst_dir\Java\JSAMLAssertionStrings.properties

     Agent_inst_dir\Java\thirdparty*

     Agent_inst_dir\Bin\smjavaagent.dll

     Agent_inst_dir\Bin\smreghost.exe

On UNIX

Agent_inst_dir/Java/soasmapi.jar

Agent_inst_dir/Java/smjavaagentapi.jar

Agent_inst_dir/Java/XmlSdkConfig.properties

Agent_inst_dir/Java/JSAMLAssertionStrings.properties

Agent_inst_dir/Java/thirdparty/*

Agent_inst_dir/Bin/smjavaagent.so

Agent_inst_dir/Bin/smreghost

Agent_inst_dir

Is the installed location of a SOA Agent for Web Servers (for example, C:\Program Files\CA\SOA Security Manager\webagent).

2. Copy your custom agent JAR file to *Agent_inst_dir*\bin.
3. Set the NETE_TXM_ROOT environment variable to specify the path to *Agent_inst_dir*\bin, the directory in which you placed your custom agent JAR file.
4. Configure the Policy Server to use the custom XML-enabled agent application.

   Note: For detailed information about configuring Agents, see the *SOA Agent Configuration Guide*.
5. Run the custom XML-enabled agent Application.

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

Java XML-Enabled Agent Functional Flow

This section describes the flow of calls required for an effective XML-enabled custom agent implemented using the Java CA SiteMinder Agent API for Java and the XML Agent Content Helper API for Java.

Force the JVM to use Xerces Parser

Include the following line at the beginning of your custom XML agent code to force the JVM to use the Xerces parser included in the SOA Security Manager SDK:

```
System.setProperty("javax.xml.parsers.DocumentBuilderFactory","org.apache.xerces.jaxp.DocumentBuilderFactoryImpl");
```

Initialize the CA SiteMinder Agent API

You initialize the CA SiteMinder Agent API by creating an AgentAPI() object and initializing connections to one or more Policy Servers by issuing the init() method. For example:

```
AgentAPI aa = new AgentAPI();
int result = aa.init (...);
...
```

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.

Once the Agent API is initialized and its connection to the Policy Server established, 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 initializing the Policy Server connection, 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, version number, and so on. The string is recorded in the Policy Server logs.

CA SiteMinder Agent API Required Code Block

The following static block of code must be included in any implementation of the CA SiteMinder Agent API class. This code should *not* be placed within a method:

```
static{
        InetAddress clientAddress;
        String clientAddressString;
        try {
                clientAddress = InetAddress.getLocalHost();
                clientAddressString = clientAddress.getHostAddress();
        }
        catch (Exception e) {
                clientAddressString = "123.123.123.123";
        }
}
```

Initialize the XML Agent Content Helper API

You must also initialize the XML Agent Content Helper API and initialize its connection to one or more Policy Servers. To do this you must first create an instance of ContentHelperService, initialize it, and then call createAgentApiService() to establish the XML Agent Content Helper API's connection to the Policy Server. For example:

```
chs = new ContentHelperService ();
String hostConfFilePath = new String ("C:/Program Files \
        /CA/SOA Security Manager/webagent/config/SmHost.conf");
chs.initialize(hostConfFilePath);
if (chs.CreateAgentApiService() != IContentHelperReturnCodes. \
        SUCCESS) {}
```

To handle policy expressions that use SOA Agent variables, you should create an instance of ServerVariablesResolver and then initialize it by setting values for those keys by calling its setVariables() method. For example:

```
// create a new Server Variables Resolver object
svr = new ServerVariablesResolver ();

// initialize XMLAgent variables and Server Variables Resolver
// object
HashMap serverVarsMap = new HashMap ();

String serverVendorKey = new String ("Microsoft");
String serverVendorValue = new String ("IIS");
serverVarsMap.put (serverVendorKey , serverVendorValue );
...

svr.setVariables (serverVarsMap );
```

Steps Required for Resource Access

Once the CA SiteMinder Agent API and XML Agent Content Helper API have been initialized, the XML-enabled agent can perform useful work. At this point it can start accepting XML documents from clients.

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 POSTed XML message

   Accept a client request to access a resource. This is the application-specific request. For example, the XML Agent would accept a Web service consumer's POST request to a URL-bound Web service.

2. Call helperFactory() to obtain an interface to a content helper class that implements the IContentHelper interface to handle XML content.

   Note: In the SOA Security Manager r6.0 SDK, the only implemented content helper is " XMLContentHelper", which handles SOAP-wrapped and raw XML.

   For example:

   ```
   IContentHelper ich
   ich = chs.helperFactory("XMLContentHelper");
   ```

3. Initialize the XML message buffer

   Write Web server-specific functions to obtain the XML message header and body and then pass these to setMessageBuffers(). This buffers the XML message for use by the XML Agent Content Helper API.

4. Identify the resource being requested

   Call identifyResource() to identify the URL of the requested resource and the rule action of the requesting client.

   Note: If the XMLResourceIdentification agent configuration parameter is set for your custom agent, the Policy Server will return an action of ProcessSOAP or ProcessXML; otherwise it will simply return POST.

5. 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.

6. Obtain the user credentials from the posted XML message

   Call gatherCredentials() to obtained the required user credentials from the posted XML message.

   Note: If your custom agent code does not call gatherCredentials(), instead manually populating the structures that gatherCredentials() typically populates, you must ensure that you set the value of the password field of the CA SiteMinder Agent API userCredentials.userCreds data structure appropriately: For XML Document Credential Collector (DCC) authentication, the value of the password field must be set to "N:" if no password is required or to "Y:password" if a password is required. The prepended "N:" or "Y:" informs SOA Security Manager whether or not to expect a password for authentication. The appropriate value is automatically supplied by gatherCredentials().

7. 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.

8. Check whether the XML message creator is authorized

   Call the CA SiteMinder Agent API authorizeEx() method to validate that the user is authorized to access the requested resource and determine whether there are unresolved variables associated with that resource.

   If authorizeEx() returns UNRESOLVED, there are XML content-based and/or XML Agent variables that must be resolved to complete authorization. Call IContentHelper.resolveVariables() to resolve XML content-based variables. If unresolved variables still remain, call ServerVariablesResolver.resolveVariables()to resolve XML Agent instance variables. Finally call authorizeEx() again to check if the message can now be authorized. (Keep repeating until all variables are resolved and authorization is successful or fails).

   Note: The CA SiteMinder Agent API authorizeEx() method is generally intended for CA internal use only, and is therefore not fully documented in the CA SiteMinder Agent API Javadoc. You should **not** attempt to use it any manner other than to authorize XML messages in XML-enabled custom agents using code similar to that shown in the following example.

   For example:

```
ResourceContextDef rcd;
RealmDef rd;
UserCredentials uc;
SessionDef ssd;
AttributeList al;
// set the transaction id
String transactionID = clientAddress +
  System.currentTimeMillis();

// create the unresolved and resolved variable buffers
StringBuffer ul = new StringBuffer ();
StringBuffer rl = new StringBuffer ();

Boolean simple = new Boolean(false);
```

```
// check to see if the user is authorized to use the resource

int azResult = AgentAPI.UNRESOLVED;

while (azResult == AgentAPI.UNRESOLVED)
{
        azResult = aa.authorizeEx (clientAddress, transactionID,
          rcd, rd, ssd, al, ul, rl, simple);

        if (azResult == AgentAPI.UNRESOLVED)
        // agent is asked to resolve variables
        {
                int xmlSdkReturn = IContentHelperReturnCodes.FAILURE;

                // try to resolve variables other than 'XML Agent'
                // variables
                xmlSdkReturn = ich.resolveVariables(ul, rl);

                if (    xmlSdkReturn == IContentHelperReturnCodes.FAILURE
                        ||
                        ul.length () > 0
                )
                // there are still variables to be resolved
                {
                        xmlSdkReturn = svr = resolveVariables(ul, rl);
                }
        }
}
// end of while (azResult == AgentAPI.UNRESOLVED)
```

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.

9. Apply any responses that modify the content of the buffered XML message

   Call applyResponses() to apply any XML message content-based responses (for example, SAML assertions) to the XML message before passing that message on to the Web service.

10. Check to see whether the buffered XML message content has changed

    Call isMessageHeaderListModified() and isMessageBodyModified() to check whether the buffered XML message has been modified as a result of applyResponses().

11. If the buffered XML message content has been changed, obtain that content from the buffer and return it to the Web server.

    Call getMessageHeaderList() and/or getMessageBody(), as applicable, to retrieve the XML message content from the agent buffers and then write Web server-specific functions to pass these back to the Web server.

12. 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.

13. Allow access to resource.

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

14. Issue a management request to update encryption keys and/or flush caches.

    (Optional) Poll the Policy Server for update commands by calling the CA SiteMinder Agent API doManagement() method. In response, encryption keys are updated for the AgentAPI() instance and/or the caches are flushed .

    If doManagement() returns an indication that encryption keys have changed, call agentKeysChanged() to force the ContentHelperService() instance to poll the Policy Server for the updated encryption keys.

    If doManagement() returns an indication of a cache update event (FLUSH_ALL or FLUSH_ALL_REALMS), call IContentHelper.flushcache() to flush the agent caches and obtain updated metadata (for example, user attributes) from the Policy Server.

Uninitialize the APIs

Once the agent is no longer needed, uninitialize all API instances by issuing the ContentHelperService.shutdown() method for each XML Agent Content Helper API instance and the unInit() method for each CA SiteMinder Agent API instance. This closes TCP connections to all policy servers.

Note: The CA SiteMinder 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 session management is required, since caching on the agent removes the need to contact the CA SOA Security Manager Policy Server to perform session validation and/or resource authorizations.