CA AuthMinder™

Java Developer's Guide

r7.1.01

technologies

Contact CA Technologies

Contact CA Support

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

- 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

Providing Feedback About Product Documentation

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

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: Getting Started

This chapter discusses the APIs provided by CA AuthMinder (later referred to as AuthMinder) Java SDK and the checks that you must perform before using the Java SDK:

This guide provides information on how to develop Web applications that use the strong and versatile modes of authentication provided by CA AuthMinder. This guide discusses Java classes and methods that you can use to programmatically integrate with AuthMinder SDK.

Note: CA AuthMinder still contains the terms Arcot and WebFort in some of its code objects and other artifacts. Therefore, you will find occurrences of Arcot and WebFort in all CA AuthMinder documentation. In addition, some of the topics in this guide do not follow the standard formatting guidelines. These inconsistencies will be fixed in a future release.

Introduction to the AuthMinder Java SDK

The AuthMinder Software Development Kit (SDK) provides a programmatic interface that includes a set of APIs to integrate with your application. It has two types of SDKs:

Authentication SDK

The AuthMinder Authentication SDK provides the APIs that can be used to authenticate the credentials supported by AuthMinder.

Issuance SDK

The AuthMinder Issuance SDK interacts with the AuthMinder Server to create, read, and update user credential information in the AuthMinder database. You can perform the following operations by using the Issuance SDK:

- Create credentials for the users
- Perform credential lifecycle management operations, such as enabling, disabling, resetting, resetting validity, resetting custom attributes, fetching, and deleting
- Manage ArcotID PKI key bag

Sample Application

The Sample Application shipped with AuthMinder serves as a code reference for the Authentication and Issuance Java SDKs. The following JSP files are shipped as a part of the AuthMinder Sample Application:

- User Operations

  Note: From 7.0 release onwards, the user operations have to be performed using Web Services that are shipped with AuthMinder. The JSP pages that the Sample Application provides is for backward compatibility.

  - ArWFCreateUser.jsp - This page is used to create the user.
  - ArWFFetchUser.jsp - This page is used to fetch the user details.
  - ArWFUpdateUser.jsp - This page is used to update the user information.
  - ArWFUserStatusChange.jsp - This page is used to enable or disable user.
  - ArWFFetchPAM.jsp - This page is used to fetch user's Personal Assurance Message (PAM).

- Credential Operations
  - ArWFCreate<*Credential*>.jsp - This page is used to create credential.
  - ArWFFetch<*Credential*>.jsp - This page is used to fetch the credential details.
  - ArWFCredStatusChange.jsp - This page is used to enable or disable user's credential.
  - ArWFReset<*Credential*>.jsp - This page is used to reset and reissue credential.
  - ArWFResetValidity.jsp - This page is used to reset the credential validity.
  - ArWFResetNotes.jsp - This page is used to reset the credential custom attributes.
  - (Only ArcotID) ArWFDownloadarcotID.jsp - This page is used to download the ArcotID PKI.
  - (Only ArcotID) ArWFSetArcotIDUnsignedAttributes.jsp - This page is used to set the unsigned attributes in ArcotID PKI.
  - (Only ArcotID) ArWFDeleteArcotIDUnsignedAttributes.jsp - This page is used to delete the unsigned attributes in ArcotID PKI.
  - (Only ArcotID) ArWFFetchArcotIDKBE.jsp - This page is used to fetch the unsigned attributes of the ArcotID PKI.
  - (Only ArcotID) ArWFAddArcotIDKBE.jsp - This page is used to add unsigned attributes to the ArcotID PKI.
  - (Only ArcotID) ArWFDeleteArcotIDKBE.jsp - This page is used to delete unsigned attributes of the ArcotID PKI.
  - ArWF<*Credential*>Authenticate.jsp - This page is used to authenticate the user with the credential.

- ArWFUPPartialPwdAuthenticate.jsp - This page is used to authenticate the users with their partial password.
- ArWFQnACallerSideAuthenticate.jsp - This page is used to authenticate the users with their QnA credential by using caller-side verification.
- ArWF<*Credential*>Synchronization.jsp - This page is used to synchronize the OTPs on the client device with the AuthMinder Server. This page is applicable for OATH OTP, ArcotID OTP-OATH, and ArcotID OTP-EMV credentials.
- (Only ArcotID) ArWFSelectArcotIDClient.jsp - This page is used to select the ArcotID PKI Client.

Note: For the operations not covered by the Sample Application, see relevant Javadocs.

AuthMinder SDK Features

This section discusses the salient features of the Authentication and Issuance SDKs.

- **SSL Support**

  You can secure the connection between the Java SDK and AuthMinder Server by using Secure Socket Layer (SSL). To set up SSL between SDK and AuthMinder Server, you must edit the AuthMinder properties files, see "Setting up SSL" (see page 138) for more information on how to do this.

- **Failover**

  Java SDKs support failover mechanism, if an instance of AuthMinder Server is not operational, then the SDKs automatically connect to any of the additional configured instances. See "Configuring Multiple AuthMinder Server Instances" (see page 137) for more information on how to do this.

- **Multiple Ways to Initialize SDK**

  You can initialize Authentication or Issuance SDKs by using either the properties file or with a map. See "Initializing the Issuance SDK" (see page 37) or "Initializing the Authentication SDK" (see page 89) for more information on how to do this.

- **Handling Multiple Operations Using Single Function**

  You can perform credential lifecycle operations on different credentials simultaneously. For example, you can create ArcotID PKI, Question and Answer, and One-Time Password credentials simultaneously using a single create() function.

- **Support for Additional Parameters**

  In addition to the mandatory inputs, the APIs also accepts additional input that can be passed as name-value pair. This input can include information, such as locale, calling application details, or profile.

Before You Begin

Before you start writing your code using the AuthMinder APIs to integrate your application with AuthMinder, ensure that:

- AuthMinder is installed and running on the required operating system.
- You have installed the correct version of the JDK required for using the AuthMinder Java SDK. See the *CA AuthMinder Installation and Deployment Guide* for this information.

Chapter 2: Understanding AuthMinder WorkFlows

AuthMinder enables you to design different workflows that can be built using the Authentication and Issuance SDKs. Based on your organization’s requirements, you can design these workflows without significantly changing the existing online experience for your users in most cases.

Note: The tasks that are listed in this chapter can be customized in multiple ways. The workflows depicted here are examples of the typical workflows. You need not follow the exact steps for each procedure mentioned in this chapter.

This chapter describes the sample workflows and provides an overview of each:

- Enrollment Workflows (see page 13)
- ArcotID PKI Authentication Workflow (see page 18)
- ArcotID PKI Roaming Download Workflow (see page 20)
- Forgot Your Password Workflow (see page 22)
- Workflow Summary (see page 23)

Enrollment Workflows

Enrollment is the process of creating a user and creating credentials for the user. The user account can either reside in the AuthMinder database or in an external directory service such as Microsoft's *Active Directory Service* (ADS) or SunOne Directory Server. If directory service is used, then user need not be created in AuthMinder, but their attributes must be mapped to the AuthMinder database attributes.

Note: See *CA AuthMinder Administration Guide* for information on how to map the user attributes from the external directory to the AuthMinder database entries.

Based on whether you are enrolling a new user or migrating existing users to AuthMinder authentication, the enrollment workflow can include:

- Enrolling New Users (see page 14)
- Migrating Existing Users (see page 14)

Enrolling New Users

To enroll new users in your system, you need to use AuthMinder Web Services. You need to call the createUserRequest message in the ArcotUserRegistrySvc Web service from your application.

Book: Refer to *CA AuthMinder Web Services Developer's Guide* for more information on creating new users.

Migrating Existing Users

AuthMinder enables you to easily migrate the users from your existing authentication method to ArcotID PKI authentication.

Note: If you are using a Directory Service (LDAP), then AuthMinder must be connected to the LDAP and you must map the LDAP attributes to the attributes supported by AuthMinder. See *CA AuthMinder Administration Guide* for more information on this.

- Migrating All Users (see page 14)
- Migrating Selected Users (see page 16)

Migrating All Users

The typical steps to migrate all users are:

1. User logs in to your application.

   The users log in to your application by using *your existing* authentication method.

2. Your application collects the information from user required to create the credential.

   Your application can either display the appropriate pages to the user. For example, you can prompt the user to set the password for ArcotID PKI or you can set the existing password as the ArcotID PKI password, and collect questions and answers if Question and Answer (QnA) is used for secondary authentication.

3. Your application invokes the create() method in the CredentialIssuance interface.

   Your application invokes the create() method in the CredentialIssuance interface to create ArcotID PKI for the user.

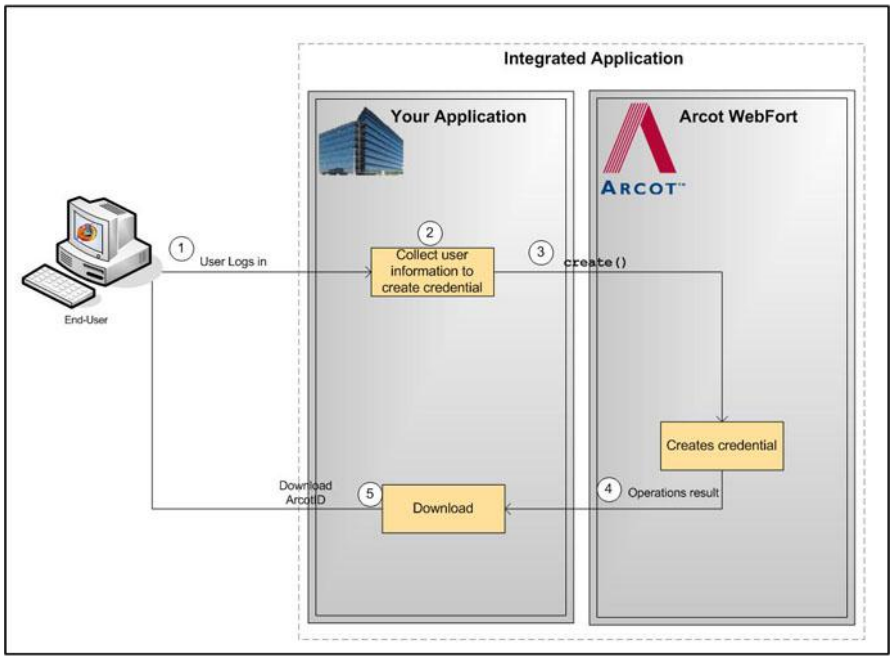
4. AuthMinder returns the result.

   If the create operation was successful, then user's ArcotID PKI is returned.

5. Application downloads the ArcotID PKI on the user's system.

   If the create() function was successful, then the application downloads the ArcotID PKI to the enduser's system without any user interaction.

The following figure illustrates the workflow for migrating all users in the system:



Migrating Selected Users

The typical steps to migrate selected users are:

1. User logs in to your application.

   The users log in to your application by using *your existing* authentication method.

2. Application gets the user status.

   Application retrieves user information and identifies whether the user account is marked for migration.

3. Application redirects user.

   Upon successful authentication, the user is redirected to migration page.

4. Your application collects the information from user required to create the credential.

   Your application can either display the appropriate pages to the user. For example, you can prompt the user to set the password for ArcotID PKI or you can set the existing password as the ArcotID PKI password, and collect questions and answers if QnA is used for secondary authentication.

5. Your application invokes the create() method in the CredentialIssuance interface.

   Your application invokes the create() method in the CredentialIssuance interface to create ArcotID PKI for the user.

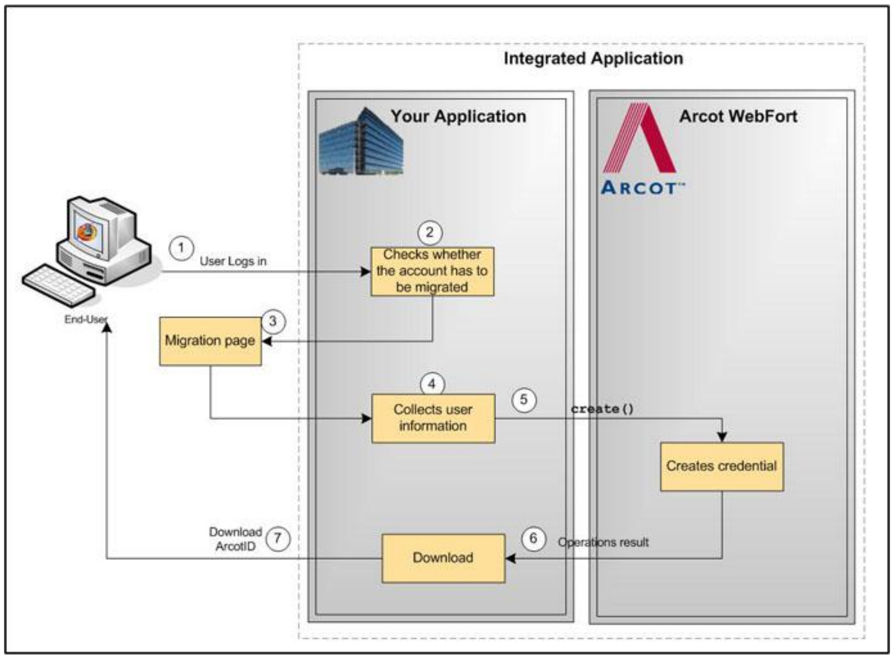
6. AuthMinder returns the result.

   If the create operation was successful, then user’s ArcotID PKI is returned.

7. Application downloads the ArcotID PKI on the user’s system.

   If the create() function was successful, then the application downloads the ArcotID PKI to the enduser’s system without any user interaction.

The following figure illustrates the workflow for migrating the users to ArcotID PKI authentication in bulk:



ArcotID PKI Authentication Workflow

During authentication, when a user specifies the credential in the authentication page, the credential is first verified by AuthMinder Server, after which the user is authenticated.

The following workflow lists the steps for ArcotID PKI authentication:

Note: In case of other credentials, refer to chapter, "Authenticating Users" (see page 89) for details of methods to invoke.

1. Application calls AuthMinder’s ArcotIDAuth.getChallenge() function.

   Your application loads the ArcotID PKI Client and makes an explicit call to the getChallenge() function in ArcotIDAuth interface. See "ArcotID PKI Authentication" (see page 93) for more information on the API.

2. User provides the credentials.

   User specifies the user name and ArcotID PKI password to log in.

3. Your application invokes the ArcotID PKI Client.

   The ArcotID PKI Client signs the challenge.

4. AuthMinder verifies the signed challenge.

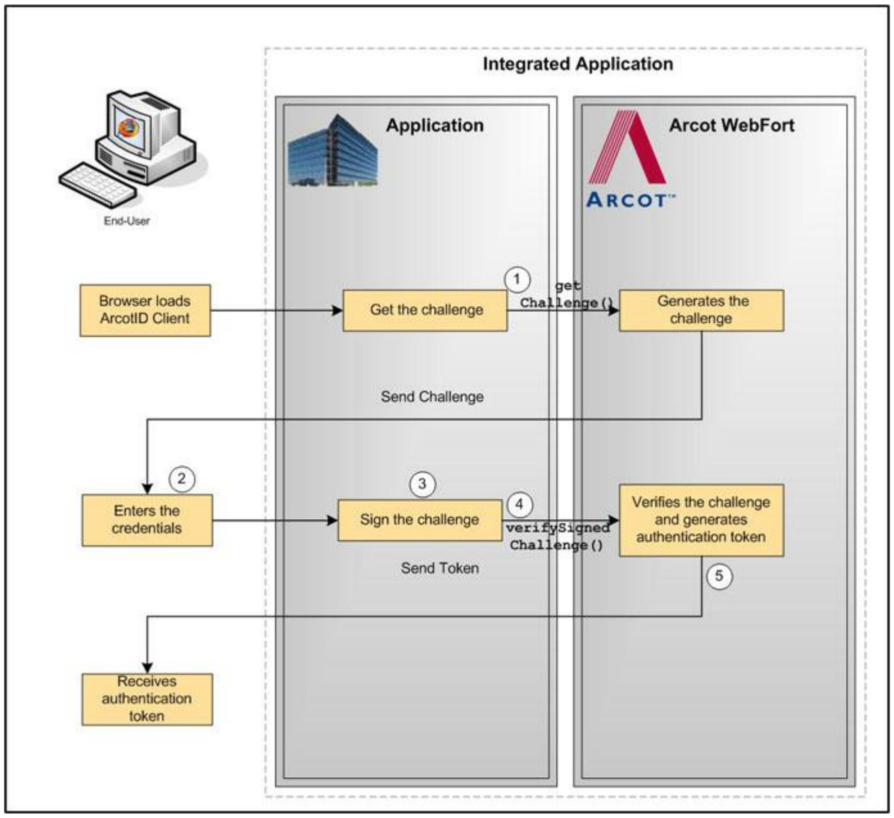
   Your application invokes the verifySignedChallenge() function in ArcotIDAuth interface to verify the challenge that is signed by using the ArcotID PKI password.

5. AuthMinder authenticates the user.

   If the verifySignedChallenge() call was successful, then the authentication token is generated and the user is authenticated successfully.

   See "Verifying the Authentication Tokens" (see page 102) for more information on the different tokens supported by AuthMinder.

The following figure illustrates the workflow for ArcotID PKI authentication process:


Integrated Application
Application
Arcot WebFort
ARCOT™
End-User
Browser loads ArcotID Client
Get the challenge
1
get Challenge()
Generates the challenge
Send Challenge
2
Enters the credentials
3
Sign the challenge
4
verifySigned Challenge()
Verifies the challenge and generates authentication token
Send Token
5
Receives authentication token

ArcotID PKI Roaming Download Workflow

To perform ArcotID PKI authentication, the ArcotID PKI of the user must be present on the user's system that is used for the current authentication session. If the user is travelling or does not have access to the system, where their ArcotID PKI is stored, then the user has to download the ArcotID PKI from the AuthMinder Server and then perform the authentication.

The typical steps for roaming download of the ArcotID PKI are:

1. User logs in to your online application.

   Your application authenticates the user.

2. User chooses to download the ArcotID PKI.

   Your application displays the appropriate page to the user to download their ArcotID PKI.

3. AuthMinder performs secondary authentication.

   Based on the secondary authentication mechanism that you are using, your application displays appropriate pages to the user. For example, you can prompt the user to:

   ■ Answer the security questions that they selected while enrolling with your application.

   ■ Enter the One-Time Password (OTP), which is sent to the user by email, SMS, or other customized method.

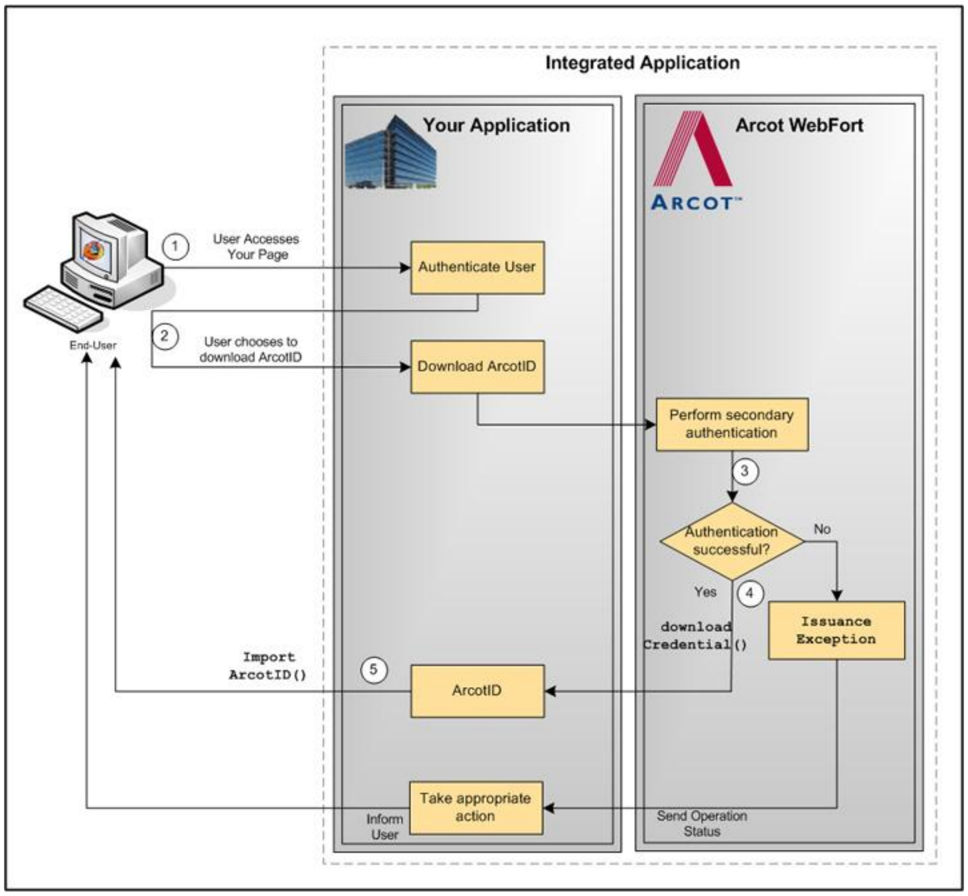
4. Your application calls AuthMinder's Issuance.Cred.downloadCredential function.

   If the secondary authentication was successful, only then your application should call the downloadCredential() function in the CredentialIssuance interface. This call downloads the corresponding ArcotID PKI to the application.

5. Download the ArcotID PKI to the user's system.

   Invoke the ImportArcotID() client-side API to download the ArcotID PKI to the enduser's system without any user interaction.

The following figure illustrates the workflow for roaming download of ArcotID PKI:


Integrated Application
Your Application
Arcot WebFort
ARCOT
End-User
1
User Accesses Your Page
Authenticate User
2
User chooses to download ArcotID
Download ArcotID
Perform secondary authentication
3
Authentication successful?
No
Yes
4
downloadCredential()
Issuance Exception
Import ArcotID()
5
ArcotID
Take appropriate action
Inform User
Send Operation Status

Forgot Your Password Workflow

If a user forgets their ArcotID PKI password, then Forgot Your Password (*FYP*) workflow can be used to reset the password.

In this method, the user is prompted to answer the questions, which they had set during enrollment or you can use any other customized method of your choice.

The typical steps for FYP workflow are:

1. User provides the user name.

   User specifies the user name to log in.

2. User clicks the FYP link.

   Because the user does not remember their password, they click the FYP link.

3. AuthMinder performs secondary authentication.

   Based on the secondary authentication mechanism that you are using, the appropriate pages are displayed to the user. For example, the user can be prompted to:

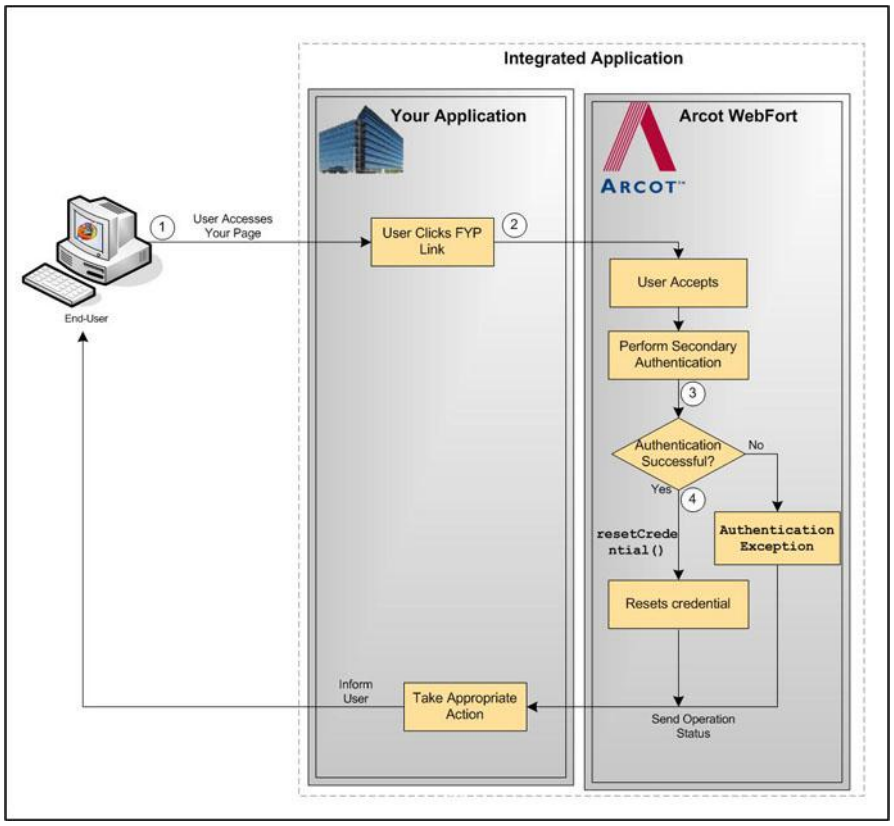
   - Answer the security questions that they selected while enrolling with your application.
   - Enter the One-Time Password (OTP), which is sent to them by email, SMS, or other customized method.

4. Your application calls AuthMinder's resetCredential() function in the CredentialIssuance interface.

   If the secondary authentication was successful, then your application must invoke the resetCredential() function in the CredentialIssuance interface. Your application prompts the user for new password and pass this as input for resetCredential() function.

   See "Resetting Credentials" (see page 54) for more information on the APIs used to reset the credential.

The following figure illustrates the Forgot Your Password workflow:



Workflow Summary

The following table provides a brief summary of the workflows that can be implemented by using the AuthMinder APIs:

Workflow	**Description**	**Dependent Workflows**
Enrollment	Creates a new user in the AuthMinder database or migrate the existing users to AuthMinder.	None
Creating the Credentials	Create the credentials for the user.	■ Enrollment
Authentication	Authenticates the user by using the credentials provided by the user.	■ Enrollment ■ Creating the Credentials

Workflow	Description	Dependent Workflows
ArcotID PKI Download	Downloads the ArcotID PKI of the user to the system.	■ Enrollment ■ Creating the Credentials
Migration	Migrates the user to ArcotID PKI authentication.	None
FYP	Resets the password.	■ Enrollment ■ Creating the Credentials

Chapter 3: Before You Use the SDK

The Authentication and Issuance SDKs constitutes a set of APIs that provide a way for your online application to programmatically integrate with AuthMinder. The AuthMinder Java SDK consists of the following components:

- The Authentication Java classes
- The Issuance Java classes
- Javadoc for the associated Java classes and methods

Note: The Sample Application shipped with AuthMinder demonstrates the usage of the Java classes and methods. See appendix, "AuthMinder Sample Application" (see page 169) for more information on AuthMinder Sample Application.

Before you use the Issuance or Authentication SDK, you must include the related JAR files in the CLASSPATH. If you are using Properties files in your application, then you must also include them in the CLASSPATH.

This chapter walks you through the steps that you must perform before you call the Issuance and Authentication API methods from your application:

- Accessing AuthMinder SDK Javadocs (see page 26)
- Adding Authentication Files in CLASSPATH (see page 26)
- Adding Issuance Files in CLASSPATH (see page 28)

Accessing AuthMinder SDK Javadocs

You can use the Javadoc provided with the AuthMinder SDK to integrate authentication and issuance services to new or existing Java applications.

If you are updating an application that is already integrated with AuthMinder, then you must refer to the Release Notes for deprecated Java APIs before making changes.

The Authentication SDK Javadoc (Arcot-WebFort-7.1.01-authentication-sdk-javadocs.zip) and the Issuance SDK Javadoc (Arcot-WebFort-7.1.01-issuance-sdk-javadocs.zip) are present in the following location:

For Windows:

<*install_location*>\Arcot Systems\docs\webfort

For Unix Platforms:

<*install_location*>/arcot/docs/webfort

Adding Authentication Files in CLASSPATH

To use the Authentication APIs, you must add its JAR files (see the following table) and the webfort.authentication.properties file in the CLASSPATH, and also ensure that the properties file is present in CLASSPATH/**properties** folder.

JAR Files

The JAR files that are required for the Authentication SDK are available in the following location:

For Windows:

<*install_location*>\Arcot Systems\sdk\client\java\lib

For Unix Platforms:

<*install_location*>/arcot/sdk/client/java/lib

File Name	**Description**
arcot/arcot-webfort-common.jar	The proprietary Java Archive (JAR) file containing the set of shared components used by Authentication SDK.
arcot/arcot-webfort-authentication.jar	Contains the Authentication APIs.

File Name	Description
external/bcprov-jdk15-146.jar	Used for cryptographic functions. For example, reading server PEM certificate.
external/commons-pool-1.5.5.jar	Used for connection pooling.

Properties File

The webfort.authentication.properties file containing the AuthMinder Server connectivity is used to initialize the Authentication SDK. You can also initialize the Authentication API through the init (see "Initializing the Authentication SDK" (see page 89)) API, which accepts the name-value pairs as input parameters.

Note: You can also copy the AuthMinder Server connectivity parameters from webfort.authentication.properties to some other file and use that file for initializing the SDK.

The properties file is available at the following location:

For Windows:

<*install_location*>\Arcot Systems\sdk\client\java\properties

For Unix Platforms:

<*install_location*>/arcot/sdk/client/java/properties

Note: If you want to edit the properties file to configure more AuthMinder Server instances and for SSL, then refer to appendix, "Additional Settings" (see page 137) for more information.

Adding Issuance Files in CLASSPATH

To use the Issuance APIs, you must add its JAR files (see the following table) and the webfort.issuance.properties file in the CLASSPATH, and also ensure that the properties file is present in CLASSPATH/**properties** folder.

JAR Files

The JAR files that are required for the Issuance SDK are available in the following location:

For Windows:

<*install_location*>\Arcot Systems\sdk\client\java\lib

For Unix Platforms:

<*install_location*>/arcot/sdk/client/java/lib

File Name	Description
arcot/arcot-webfort-common.jar	The proprietary Java Archive (JAR) file containing the set of shared components used by Issuance SDK.
arcot/arcot-webfort-issuance.jar	The JAR file containing the Issuance APIs.
external/bcprov-jdk15-146.jar	Used for cryptographic functions. For example, reading server PEM certificate.
external/commons-pool-1.5.5.jar	Used for connection pooling.

Properties Files

The webfort.issuance.properties file containing the AuthMinder Server connectivity is used to initialize the Issuance SDK. You can also initialize the Issuance API through the init API (see "Initializing the Issuance SDK" (see page 37)), which accepts the name-value pairs as input parameters.

Note: You can also copy the AuthMinder Server connectivity parameters from webfort.issuance.properties to some other file and use that file for initializing the SDK.

The properties file is available at the following location:

For Windows:

<*install_location*>\Arcot Systems\sdk\client\java\properties

For Unix Platforms:

<*install_location*>/arcot/sdk/client/java/properties

Note: If you want to edit the properties file to configure more AuthMinder Server instances and for SSL, then refer to appendix, "Additional Settings" (see page 137) for more information.

Chapter 4: Managing Users

Creating a new user in AuthMinder is a one-time operation, performed only when a new user is to be added to AuthMinder database.

In previous releases, the Issuance Java API provided a programmable interface, which could be used by Java clients (such as Java Servlets and JSP pages) to send user management requests to AuthMinder Server. In release 7.0, the user management interfaces and methods have been deprecated from the Issuance API (**Issuance**). Now, you must use the User Management Web service (**ArcotUserRegistrySvc**) for the purpose. This Web service creates a request message that is sent to the AuthMinder Server, receives the response, and packages it as return response to be read by your application.

Book: Refer to the section, "Performing User Operations" in *Chapter 6,* "Managing Users and Accounts" in the *CA AuthMinder Web Services Developer's Guide* for more information on how to use the User Management Web service for creating and managing users in AuthMinder. This chapter provides an overview of how to use the Web service to create users in AuthMinder database and operations it provides.

Chapter 5: Integrating ArcotID PKI Client with Application

The **ArcotID PKI Client** is a software that is used by the end user to sign the challenge provided by the AuthMinder Server. If you are planning to implement ArcotID PKI-based authentication, then you must integrate ArcotID PKI Client with application before you call ArcotID PKI authentication APIs. This chapter provides information on different client types, details on how to integrate them with application, and lists the APIs provided by ArcotID PKI Client. It covers the following topics:

ArcotID PKI Client Overview

The ArcotID PKI Client is used for signing the AuthMinder-issued challenge at the user end, but it also facilitates the download of the user's ArcotID PKI. To support a wide variety of end user environments, the ArcotID PKI Client is available as a Flash client and as a signed Java applet. Each client type offers different levels of convenience and capabilities. The degree of user interaction and administration rights required for configuration vary depending on the client selected.

Flash Client

This implementation of ArcotID PKI Client runs in any Web browser that has Adobe Flash Player (version 9 or higher) installed.

Note: If you are using ArcotID PKI Flash Client for ArcotID PKI operations, then the application serving the Flash client must be enabled for HTTPS.

Signed Java Applet

This implementation of the ArcotID PKI Client can run in any Web browser that has Sun Java Runtime Environment (JRE) installed.

Note: When using the CA signed Java applet, the user will be presented with a security message that requires the user to accept the signed applet before it is invoked.

Copying ArcotID PKI Client Files

ArcotID PKI Client is an end-user system component. Therefore based on the client type that you are planning to use, you must package the relevant files to the correct locations on the system where the application is running.

This section discusses the files that needs to be packaged with the application:

- For Flash Client (see page 34)
- For Java Signed Applet (see page 35)

For Flash Client

The Flash client package contains the following files:

- arcotclient.js

  Contains the ArcotID PKI Flash Client APIs.

- ArcotIDClient.swf

  Contains the ArcotID PKI Flash Client implementation.

To configure a Flash Client:

1. Copy arcotclient.js and ArcotIDClient.swf files to an appropriate directory within your application home.
2. Include the following JavaScript code in the Web page of your application from where the APIs will be invoked:
   ```
   <script type="text/javascript"
   src="location_to_arcotclient.js"></script>
   ```
   In the preceding code snippet, replace location_to_arcotclient.js with the path to arcotclient.js.
3. Ensure that in the all application pages, ArcotIDClient.swf is referred with same URL.

For Java Signed Applet

The Java Signed Applet client package contains the following files:

- arcotclient.js

  Contains the Java Signed Applet client APIs.

- ArcotApplet.jar (for Sun JRE)

  Contains the Java Signed Applet client implementation.

To configure the Java Signed Applet Client:

1. Copy arcotclient.js and ArcotApplet.jar to an appropriate directory within your application home.
2. Include the following JavaScript code in the relevant Web page of your application:
   ```
   <script type="text/javascript"
   src="location_to_arcotclient.js"></script>
   ```
   In the preceding code snippet, replace location_to_arcotclient.js with the path to arcotclient.js.
3. Ensure that in the all application pages, the Java Applet is referred with same URL.

ArcotID PKI Client APIs

If you are implementing ArcotID PKI authentication, then your application must integrate with ArcotID PKI Client APIs for:

- Downloading ArcotID PKI (see page 36)
- Signing the Challenge (see page 36)

Downloading ArcotID PKI

To download the ArcotID PKI from the application to the end-user system, you must use the ImportArcotID() function. This function takes the base-64 encoded string of the ArcotID PKI that has to be downloaded and the storage mode as the input parameters.

The ArcotID PKI can be temporarily downloaded for the current session or can be downloaded permanently. This storage mode is specified by the storage medium selected for storing the ArcotID PKI. An ArcotID PKI can be stored in any of the following:

- Hard Disk
- Universal Serial Bus (USB)
- Memory

The downloaded ArcotID PKI is saved with the .aid extension. The name of the ArcotID PKI file is derived from the hash value of user name, organization name, and domain name.

Signing the Challenge

The challenge from the AuthMinder Server must be signed by using the SignChallengeEx() function of the client API.

Book: Refer to *ArcotID Client Reference Guide* for more information on the API details.

Chapter 6: Performing Issuance Operations

For AuthMinder to authenticate users, an account for each user has to be created in the database, as discussed in the chapter, "Managing Users" (see page 31). Then you need to create credentials for users.

This chapter provides description of the APIs that are used for credential operations. It covers the following topics:

Initializing the Issuance SDK

Initialize the Issuance SDK by using the Issuance class in the com.arcot.webfort.issuance.api package. After initialization, it returns an appropriate message to the calling application.

The Issuance class provides two methods to initialize the Issuance SDK.

Method 1: Initializing the SDK by Using the Map

This method initializes the Issuance Application based on the map provided. The following table provides the details of the init() method:

Description	**Input Values**	**Output Value**
Initializes the Issuance SDK by using the provided map.	■ map The key-value pair specifying the configuration information. The map values are same as the parameters listed in the webfort.issuance.properties file. **Book:** Refer to appendix, "Configuration Files and Options" in the *CA AuthMinder Installation and Deployment Guide* for more information on the parameters in this properties file. ■ locale The locale of the API. The default value is set to en_US.	Returns an exception if the SDK is not initialized successfully.

Method 2: Initializing the SDK by Using the Properties File

This method initializes the Issuance SDK by using the parameters listed in the properties file. If you pass NULL, then the parameters are read from the webfort.issuance.properties file. If you provide a different file name containing these configuration parameters, then that file is read instead.

The following table provides the details of the initialization method using the properties method:

Description	**Input Values**	**Output Value**
Initializes the Issuance SDK by using the properties file.	■ location The absolute path of the properties file. By default, this file is available at *<install_location>*/sdk/client/java/properties ■ locale The locale of the API. The default value is set to en_US.	Returns an exception if the SDK is not initialized successfully.

Releasing the Issuance SDK Resources

The Issuance class also provides a method to release the resources such as sockets that are used by Issuance SDK.

Important! This method must be invoked before re-initializing the SDK.

The following table provides the details of the release() method:

Description	**Input Values**	**Output Value**
Releases the Issuance SDK.	The locale of the API.	Returns an exception if the API is not released successfully.

Before You Proceed

The Issuance SDK performs the user status checks (if enabled) before performing the credential operations that are discussed in this chapter. This section lists these user status checks, supported credential states, supported transitions between the credential states, and the credential operations that are possible on a particular credential state. It covers the following topics:

- Checking the User Status (see page 39)
- Credential States and Supported Transitions (see page 40)
- Credential Operations and States (see page 41)

Checking the User Status

AuthMinder uses the user status information *before* performing some of the credential operations. A user's status in the database can be either INITIAL, ACTIVE, INACTIVE, or DELETED.

Note: For Issuance SDK to perform these checks, you must enable this option when you create configurations using the Administration Console. Refer to the section, "Credential Profiles and Policies" in Chapter 5, "Managing Global AuthMinder Configurations" in the *CA AuthMinder Administration Guide* for more information.

The following table lists all the credential operations and the user checks that are performed depending on the type of operation:

Operation	**Checks**		
	User Existence	**User Status**	**User Attribute**
Create	Yes	Yes	Yes
Delete	No	No	No
Disable	No	No	No
Enable	Yes	Yes	No
Fetch	No	No	No
Fetch QnA Configuration	No	No	No
Reissue	Yes	Yes	No
Reset	Yes	Yes	No
Reset Custom Attributes	Yes	Yes	No
Reset Validity	Yes	Yes	No
Download Credential	Yes	Yes	No
Delete Unsigned Attributes	No	No	No
Set Unsigned Attributes	No	No	No
Add ArcotID Key Bag Elements	No	No	No
Fetch ArcotID Key Bag Elements	No	No	No
Delete ArcotID Key Bag Elements	No	No	No

Credential States and Supported Transitions

AuthMinder supports the following states for a credential that is issued to a user:

- **ACTIVE**

  Indicates that the credential is active and can be used for authentication.

- **DISABLED**

  The credential is disabled by the administrator.

- **LOCKED**

  The credential is locked when the user consecutively fails to authenticate for the maximum number of negative attempts configured. For example if the maximum attempts configured is 3, then the third attempt with wrong credential will lock the credential.

- **VERIFIED**

  The credential is verified when the OTP submitted by the user is authenticated by the AuthMinder Server successfully.

  Note: This status is applicable only for OTP.

- **DELETED**

  The credential of the user is deleted.

When you perform an operation on a credential, the status of the credential might be changed after the operation is performed successfully on the credential. For example, when the user successfully authenticates with their OTP, then status of the user's OTP is changed to VERIFIED.

The following table lists the transitions possible between the supported credential states:

Current State	**Change State to**				
	Enabled	**Locked**	**Disabled**	**Deleted**	Verified **(for OTP *only*)**
Active	Yes	Yes	Yes	Yes	Yes
Locked	Yes	Yes	Yes	Yes	No
Disabled	Yes	No	Yes	Yes	No
Deleted	No	No	No	Yes	No
Verified	No	No	No	Yes	No

Credential Operations and States

The following table lists all credential operations and whether each operation is allowed on a specific state of the credential. If the state of the credential changes after an operation, then the table also provides the next state of the credential.

Note: *Allowed* indicates that the operation can be performed, but the state of the credential will not change after the operation.

Operation	**State**				
	Enabled	**Locked**	**Disabled**	**Deleted**	**Verified (for OTP only)**
Create	Not allowed	Not allowed	Not allowed	Allowed -> Enabled	Not applicable
Enable	Allowed -> Enabled	Allowed -> Enabled	Allowed -> Enabled	Not allowed	Not applicable
Disable	Allowed -> Disabled	Allowed -> Disabled	Allowed -> Disabled	Not allowed	Not applicable
Fetch	Allowed	Allowed	Allowed	Allowed	Allowed
FetchQnAConfiguration	Allowed	Allowed	Allowed	Allowed	Not applicable
Reset	Allowed -> Enabled	Allowed -> Enabled	Allowed -> Enabled	Not allowed	Not applicable
Reset Validity	Allowed	Allowed	Allowed	Not allowed	Not applicable
Download Credential	Allowed	Allowed	Allowed	Not allowed	Not applicable
Reset Custom Attributes	Allowed	Allowed	Allowed	Not allowed	Not applicable
Reissue	Allowed -> Enabled	Allowed -> Enabled	Allowed -> Enabled	Not allowed	Not applicable

Operation	**State**				
	Enabled	**Locked**	**Disabled**	**Deleted**	**Verified (for OTP only)**
Delete Unsigned Attributes (for ArcotID *only*)	Allowed	Allowed	Allowed	Not allowed	Not applicable
Set Unsigned Attributes (for ArcotID *only*)	Allowed	Allowed	Allowed	Not allowed	Not applicable
Add ArcotID Key Bag Elements	Allowed	Allowed	Allowed	Not allowed	Not applicable
Fetch ArcotID Key Bag Elements	Allowed	Allowed	Allowed	Not allowed	Not applicable
Delete ArcotID Key Bag Elements	Allowed	Allowed	Allowed	Not allowed	Not applicable
Delete	Allowed -> Deleted	Allowed -> Deleted	Allowed -> Deleted	Not allowed	Not applicable

Credential Operations

This section describes the credential lifecycle operations that are supported by the Issuance API. The operations listed in this chapter can be performed on all credentials that are supported by AuthMinder, and can be performed by using any of the following method:

- **By using AuthMinder SDKs**

  This mode enables you to automate the credential management operations programmatically.

- **By using Administration Console**

  Administration Console is a Web-based application and is typically suitable for Customer Support Representatives (CSRs), who handle the user requests (such as, disabling the credential, enabling the credential, or resetting the credential validity.)

  Note: Refer to *CA AuthMinder Administration Guide* for more information on using the Administration Console.

This section covers the following credential lifecycle operations:

- Preparing Additional Input (see page 45)
- Preparing the Input (see page 46)
- Creating Credentials (see page 51)
- Disabling Credentials (see page 52)
- Enabling Credentials (see page 53)
- Resetting Credentials (see page 54)
- Fetching Credential Details (see page 55)
- Downloading Credentials (see page 56)
- Reissuing Credentials (see page 57)
- Resetting Credential Validity (see page 58)
- Resetting Custom Attributes (see page 59)
- Fetching QnA Configuration (see page 60)
- Adding Elements to ArcotID PKI Key Bag (see page 61)
- Fetching ArcotID PKI Key Bag Elements (see page 62)
- Deleting ArcotID PKI Key Bag Elements (see page 63)
- Setting Unsigned Attributes (see page 64)
- Deleting Unsigned Attributes (see page 65)

Note: Each operation discussed in this chapter can be performed simultaneously by using different credentials. If the operation fails for a single credential, then the operations for other credentials are also considered invalid. For example, if you are creating ArcotID PKI, QnA, and OTP, and the ArcotID PKI and OTP creation was successful, while the QnA creation failed, then all the three credentials have to be created again.

Preparing Additional Input

You need to prepare additional inputs if you plan to augment AuthMinder's standard authentication capability by implementing plug-ins, then you need to set the extra information that must be sent to AuthMinder Server in name-value pairs. AuthMinder's com.arcot.webfort.common.api provides you the AdditionalInput class, which enables you to set this additional information that you plan to use.

Some of the pre-defined additional input parameters supported by the AdditionalInput class are:

- AR_WF_LOCALE_ID

  Specifies the locale that AuthMinder will use in returning the messages back to the calling application.

- AR_WF_CALLER_ID

  This is useful in tracking transactions. You can use session ID or transaction ID for specifying this information.

- AR_WF_OTP_TXN_SIGN_DATA

  Specifies the transaction data that the end user enters in the Challenge field of the ArcotID PKI OTP client to generate a passcode in the Sign mode. The maximum length of the signed data is 64 bytes. This implementation of the Transaction Signing feature conforms to the OATH Challenge-Response Algorithm (OCRA) as defined by RFC 6287.

- AR_WF_TXN_FILE_LOG_TRACE

  Enables TRACE logging for the transaction. The presence of the identifier irrespective of the value enables TRACE logging.

- AR_WF_TXN_FILE_LOG_LEVEL

  Used to control the log level for the transaction. The supported values are:

  - 1 for WARNING
  - 2 for INFO
  - 3 for DETAIL

  See appendix, "AuthMinder Logging" (see page 125) for more information on log levels.

- AR_WF_TXN_LOG_SENSITIVE_DATA

  Indicates whether the sensitive data must be logged for the current transaction. For example, USERNAME of the user. The presence of the identifier irrespective of the value enables this logging.

- AR_WF_TXN_DB_LOG_QUERY_DETAILS

  Indicates whether the database query execution has to be logged in detail. The presence of the identifier irrespective of the value enables this logging.

Preparing the Input

Preparing the input for this interface and sub-interface includes preparing:

- Common Input (see page 46)
- Credential-Specific Input (see page 48)

Common Input

The CredentialInput interface provides the common configurations to all the credential types. The following information is set by using this interface:

- Validity
- Custom Attributes
- Profile Name
- Disable Period

Validity

The Issuance API enables you to set a period for which the credential will be valid. Invoke the setValidity() method of the CredentialInput class if you want to pass a specific calendar date, or use setValidityEx() class if you want to use the ArcotDate.Type Class to set the validity date.

The validity of the credential is taken as input by resetValidity() methods.

Custom Attributes

The Issuance API enables you to add custom attributes for each credential type. This feature helps you to maintain any additional credential information. For example, if you do not want the user to download their ArcotID PKI on more than five systems, then you can create an attribute with this information. This is taken as input by create() or resetNotes() methods.

To add custom attributes, invoke the setNote() method of the CredentialInput class.

Profile Name

Typically, same set of credential information could well be applied to many users. In such cases, to avoid the cumbersome task of entering the credentials for each user individually, you can create a *profile* with all common information and share this profile among multiple users. Each profile is identified by a unique *Profile Name*.

The Issuance API enables you to set the profile name for the credential. To set the profile name, invoke the setProfileName() method of the CredentialInput class.

Note: If the profile is not set, then the default profile for the credential is used.

Disable Period

If your users want to go on a vacation or on long leave, then their credentials can be disabled only for that period, after which the credentials will be enabled automatically. This feature facilitates credential activation without the user making a request to User Administrator (UA) to do so.

The Issuance API enables you to set the disable period for the credential. To set the disable period, invoke the setDisableStartTime() and setDisableEndTime() methods of the CredentialInput class. The setDisableStartTime() and setDisableEndTime() methods use the ArcotDate.Type Class for setting the disable period.

ArcotDate.Type Class

The ArcotDate.Type class enables you to set the validity, disable start and end periods by using the following date formats:

- **Current Date**

  Uses the current date of the AuthMinder Server to set the validity or disable period.

- **Forever Date**

  Specifies the credential will be valid forever and will not expire.

- **Relative Date**

  Uses a relative date corresponding to the disable start date. For example, if the relative date is one month, then disable end date would be one month after the disable start date.

- **Specific Date**

  Uses a date that is specified by your application to set the validity or disable period.

Credential-Specific Input

The com.arcot.webfort.issuance.api package provides the interface that you can use to set the information specific to the following supported credentials:

- Preparing ArcotID PKI Input
- Preparing QnA Input
- Preparing Password Input
- Preparing OATH OTP Input
- Preparing ArcotID OTP Input
- Preparing EMV OTP Input

Preparing ArcotID PKI Input

The following ArcotID PKI inputs can be set by using the ArcotIDInput class:

1. **Unsigned Attributes**

   You can define ArcotID PKI attributes while or after creating an ArcotID PKI for the user. Such attributes are called *unsigned attributes* because these attributes (name-value pairs) are set in the unsigned portion of the ArcotID PKI.

   Note: If you add an attribute that already exists, then the current attributes will be overwritten by the new value.

   To set unsigned attributes:

   a. Use the ArcotIDInput class to obtain the methods that set the information of the ArcotID PKI.

   b. Use ArcotIDAttribute class to define the unsigned attributes to set in the ArcotID PKI.

   c. Invoke the setUnsignedAttributes method in the ArcotIDInput class.

2. **Password**

   To set the password for the ArcotID PKI or change the current ArcotID PKI password, you must use the setPassword method. To set the ArcotID PKI password:

   a. Use the ArcotIDInput class to obtain the methods that set the information of the ArcotID PKI.

   b. Invoke the setPassword method in the ArcotIDInput class.

3. **ArcotID PKI Attributes**

   To fetch ArcotID PKI attributes in ArcotIDResponse, you must enable the setFetchAttributeFlag() flag. To fetch ArcotID PKI attributes:

   a. Use the ArcotIDInput class to obtain the methods that set the information of the ArcotID PKI.

b. Invoke the setFetchAttribute method in the ArcotIDInput class.

Preparing QnA Input

The following QnA inputs can be set by using the QnAInput class:

1. **Set Questions and Answers**

   The questions and answers for the QnA authentication must be set by using the QnAInput class. To add the questions and answers:

   a. Use the QnAInput class to obtain the methods that set the information of QnA.

   b. Invoke the setQuestionAnswer method in the QnAInput class.

2. **Update Questions and Answers**

   The questions and answers for the QnA authentication must be updated by using the QnAInput class. To update the questions and answers:

   a. Use the QnAInput class to obtain the methods that set the information of QnA.

   b. Invoke the updateQuestionAnswer method in the QnAInput class. You can update the following:

      ■ Add questions and answers

      ■ Change answers of questions

      ■ Change questions

      ■ Delete questions

Preparing Password Input

The password for the password authentication is set by using the UPInput class. To set the password:

1. Use the UPInput class to obtain the methods that set the information of password.
2. Invoke the setPassword method in the UPInput class.

Preparing OATH OTP Input

The following OATH OTP inputs can be set by using the OATHOTPInput class:

1. **Token Identifier**

   To set the token ID that is used to issue the OATH OTP you must use the OATHOTPInput class. To set the token ID:

   a. Use the OATHOTPInput class to obtain the methods that set the token ID of the OTP.
   b. Invoke the setTokenID method in the OATHOTPInput class.

2. **Reuse Token**

   To set if the abandoned token can be reused, you must use the OATHOTPInput class. To reuse the token:

   a. Use the OATHOTPInput class to obtain the methods that set the token ID of the OTP.
   b. Invoke the setReUseToken method in the OATHOTPInput class.

Preparing ArcotID OTP Input

To set or change the password that is used to generate ArcotID OTP, you must use the setPassword method, as follows:

1. Use the ArcotOTPInput class to obtain the methods that set the information of the ArcotID PKI.
2. Invoke the setPassword method in the ArcotOTPInput class.

Preparing EMV OTP Input

EMV OTPs are compliant to *Europay MasterCard VISA* (EMV) standards. To set or change the password that is used to generate EMV OTP, you must use the setPassword method, as follows:

1. Use the EMVInput class to obtain the methods that set the information of the EMV OTP.
2. Invoke the setPassword method in the EMVInput class.

Creating Credentials

The com.arcot.webfort.issuance.api package provides the CredentialIssuance interface that contains the methods to create the credentials for the user.

To create credentials:

1. Depending on the type of credential you want to create, use the respective <*CredentialName*>Input class to obtain an object that implements the class.

   The input required for each credential is different. For example, password is needed for password as well as ArcotID PKI credential, while questions and corresponding answers are required for QnA credentials.

   Note: See "Credential Operations Summary" (see page 69) for the input details required by different credentials.

2. Use the CredentialInput abstract class to obtain the methods that set the common information of the credential.

3. Invoke the CredentialInputList class to pass the input classes of different credentials.

4. (**Optional**) If you are implementing a plug-in, then invoke the setAdditionalInput() method in the AdditionalInput class to fill the AdditionalInput.

   This class provides the additional information that is set as a name-value pair. See "Preparing Additional Input" (see page 45) for more information.

5. Invoke the create() method of the CredentialIssuance interface to create the credentials.

   This method returns an instance of the CredentialResponse interface, which specifies the details of all credentials and the transaction.

Handling Errors

Exceptions are returned for any errors that occurred while executing any of the Issuance API methods. The com.arcot.webfort.issuance.api.exception and com.arcot.webfort.common.api.exception packages contain these exceptions. See "Issuance Exceptions" (see page 146) and "Common Exceptions" **(see page 145)** for more information on exception classes.

Note that if no exception is thrown, then processing was successful and the return object can be referenced for processing results. The lack of an error does not necessarily mean that the request was successful.

Disabling Credentials

User credentials can be disabled for a specified time interval. For example, if an employee goes for long vacation, then the credentials of this user can be disabled to prevent any unauthorized access during their absence.

To disable credentials:

1. Depending on the type of credential you want to disable, use the respective **<***CredentialName*>Input class to obtain an object that implements the class.

   The input required for each credential is different. For example, password is needed for password as well as ArcotID PKI credential, while questions and corresponding answers are required for QnA credentials.

   Note: See "Credential Operations Summary" (see page 69) for the input details required by different credentials.

2. Use the CredentialInput abstract class to obtain the methods that set the common information of the credential.

3. Invoke the CredentialInputList class to pass the input classes of different credentials.

4. (**Optional**) If you are implementing a plug-in, then invoke the setAdditionalInput() method in the AdditionalInput class to fill the AdditionalInput.

   This class provides the additional information that is set as a name-value pair. See "Preparing Additional Input" (see page 45) for more information.

5. Invoke the disable() method of the CredentialIssuance interface to disable the credentials.

   This method returns an instance of the CredentialResponse interface, which specifies the details of all credentials and the transaction.

Handling Errors

Exceptions are returned for any errors that occurred while executing any of the Issuance API methods. The com.arcot.webfort.issuance.api.exception and com.arcot.webfort.common.api.exception packages contain these exceptions. See "Issuance Exceptions" (see page 146) and "Common Exceptions" **(see page 145)** for more information on exception classes.

Note that if no exception is thrown, then processing was successful and the return object can be referenced for processing results. The lack of an error does not necessarily mean that the request was successful.

Enabling Credentials

The enable method is used to activate the disabled or locked credential of a user. For example, a credential can be disabled or locked if a user tries to authenticate by using the wrong credential or exceeds the configured maximum number of allowed attempts.

To enable a credential:

1. Depending on the type of credential you want to enable, use the respective **<***CredentialName*>Input class to obtain an object that implements the class.

   The input required for each credential is different. For example, password is needed for password as well as ArcotID PKI credential, while questions and corresponding answers are required for QnA credentials.

   Note: See "Credential Operations Summary" (see page 69) for the input details required by different credentials.

2. Use the CredentialInput abstract class to obtain the methods that set the common information of the credential.

3. Invoke the CredentialInputList class to pass the input classes of different credentials.

4. (**Optional**) If you are implementing a plug-in, then invoke the setAdditionalInput() method in the AdditionalInput class to fill the AdditionalInput.

   This class provides the additional information that is set as a name-value pair. See "Preparing Additional Input" (see page 45) for more information.

5. Invoke the enable() method of the CredentialIssuance interface to enable the credentials.

   This method returns an instance of the CredentialResponse interface, which specifies the details of all credentials and the transaction.

Handling Errors

Exceptions are returned for any errors that occurred while executing any of the Issuance API methods. The com.arcot.webfort.issuance.api.exception and com.arcot.webfort.common.api.exception packages contain these exceptions. See "Issuance Exceptions" (see page 146) and "Common Exceptions" **(see page 145)** for more information on exception classes.

Note that if no exception is thrown, then processing was successful and the return object can be referenced for processing results. The lack of an error does not necessarily mean that the request was successful.

Resetting Credentials

The Issuance APIs enables you to reset the credential. For example, you can reset the ArcotID PKI password or questions and answers.

To reset the credential:

1. Depending on the type of credential you want to reset, use the respective <*CredentialName*>Input class to obtain an object that implements the class.

   The input required for each credential is different. For example, password is needed for password as well as ArcotID PKI credential, while questions and corresponding answers are required for QnA credentials.

   Note: See "Credential Operations Summary" (see page 69) for the input details required by different credentials.

2. Use the CredentialInput abstract class to obtain the methods that set the common information of the credential.
3. Invoke the CredentialInputList class to pass the input classes of different credentials.
4. (**Optional**) If you are implementing a plug-in, then invoke the setAdditionalInput() method in the AdditionalInput class to fill the AdditionalInput.

   This class provides the additional information that is set as a name-value pair. See "Preparing Additional Input" (see page 45) for more information.

5. Invoke the resetCredential() method of the CredentialIssuance interface to reset the credential.

   This method returns an instance of the CredentialResponse interface, which specifies the details of all credentials and the transaction.

Handling Errors

Exceptions are returned for any errors that occurred while executing any of the Issuance API methods. The com.arcot.webfort.issuance.api.exception and com.arcot.webfort.common.api.exception packages contain these exceptions. See "Issuance Exceptions" (see page 146) and "Common Exceptions" **(see page 145)** for more information on exception classes.

Note that if no exception is thrown, then processing was successful and the return object can be referenced for processing results. The lack of an error does not necessarily mean that the request was successful.

Fetching Credential Details

To read the details of the user credentials, you need to implement the fetch() method.

To read a user's credential information:

1. Depending on the type of credential whose details have to be fetched, use the respective **<***CredentialName*>Input class to obtain an object that implements the class.

   The input required for each credential is different. For example, password is needed for password as well as ArcotID PKI credential, while questions and corresponding answers are required for QnA credentials.

   Note: See "Credential Operations Summary" (see page 69) for the input details required by different credentials.

2. Use the CredentialInput abstract class to obtain the methods that set the common information of the credential.
3. Invoke the CredentialInputList class to pass the input classes of different credentials.
4. (**Optional**) If you are implementing a plug-in, then invoke the setAdditionalInput() method in the AdditionalInput class to fill the AdditionalInput.

   This class provides the additional information that is set as a name-value pair. See "Preparing Additional Input" (see page 45) for more information.

5. Invoke the fetch() method of the CredentialIssuance interface to read the credential details.

   This method returns an instance of the CredentialResponse interface, which specifies the details of all credentials and the transaction.

Handling Errors

Exceptions are returned for any errors that occurred while executing any of the Issuance API methods. The com.arcot.webfort.issuance.api.exception and com.arcot.webfort.common.api.exception packages contain these exceptions. See "Issuance Exceptions" (see page 146) and "Common Exceptions" **(see page 145)** for more information on exception classes.

Note that if no exception is thrown, then processing was successful and the return object can be referenced for processing results. The lack of an error does not necessarily mean that the request was successful.

Downloading Credentials

To download the credentials of the users to their system, you need to implement the downloadCredential() method. This method is used for ArcotID PKI, ArcotID OTP-OATH, and ArcotID OTP-EMV credentials.

To download the credential information:

1. Depending on the type of credential whose details have to be fetched, use the respective **<***CredentialName*>Input class to obtain an object that implements the class.

   The input required for each credential is different. For example, password is needed for password as well as ArcotID PKI credential, while questions and corresponding answers are required for QnA credentials.

   Note: See "Credential Operations Summary" (see page 69) for the input details required by different credentials.

2. Use the CredentialInput abstract class to obtain the methods that set the common information of the credential.

3. Invoke the CredentialInputList class to pass the input classes of different credentials.

4. (**Optional**) If you are implementing a plug-in, then invoke the setAdditionalInput() method in the AdditionalInput class to fill the AdditionalInput.

   This class provides the additional information that is set as a name-value pair. See "Preparing Additional Input" (see page 45) for more information.

5. Invoke the downloadCredential() method of the CredentialIssuance interface to download the credentials.

   This method returns an instance of the CredentialResponse interface, which specifies the details of all credentials and the transaction.

Handling Errors

Exceptions are returned for any errors that occurred while executing any of the Issuance API methods. The com.arcot.webfort.issuance.api.exception and com.arcot.webfort.common.api.exception packages contain these exceptions. See "Issuance Exceptions" (see page 146) and "Common Exceptions" **(see page 145)** for more information on exception classes.

Note that if no exception is thrown, then processing was successful and the return object can be referenced for processing results. The lack of an error does not necessarily mean that the request was successful.

Reissuing Credentials

The Issuance API enables you to re-create the credentials for the user. If the credential has been reissued for the user, then the user cannot log in by using their old credential.

To reissue a credential:

1. Depending on the type of credential you want to reissue, use the respective <*CredentialName*>Input class to obtain an object that implements the class.

   The input required for each credential is different. For example, password is needed for password as well as ArcotID PKI credential, while questions and corresponding answers are required for QnA credentials.

   Note: See "Credential Operations Summary" (see page 69) for the input details required by different credentials.

2. Use the CredentialInput abstract class to obtain the methods that set the common information of the credential.

3. Invoke the CredentialInputList class to pass the input classes of different credentials.

4. (**Optional**) If you are implementing a plug-in, then invoke the setAdditionalInput() method in the AdditionalInput class to fill the AdditionalInput.

   This class provides the additional information that is set as a name-value pair. See "Preparing Additional Input" (see page 45) for more information.

5. Invoke the reissue() method of the CredentialIssuance interface to recreate the credential.

   This method returns an instance of the CredentialResponse interface, which specifies the details of all credentials and the transaction.

Handling Errors

Exceptions are returned for any errors that occurred while executing any of the Issuance API methods. The com.arcot.webfort.issuance.api.exception and com.arcot.webfort.common.api.exception packages contain these exceptions. See "Issuance Exceptions" (see page 146) and "Common Exceptions" **(see page 145)** for more information on exception classes.

Note that if no exception is thrown, then processing was successful and the return object can be referenced for processing results. The lack of an error does not necessarily mean that the request was successful.

Resetting Credential Validity

Issued credentials are valid for the period that is specified at the time they are created. The CredentialIssuance interface provides resetValidity() method, which helps to reset the validity period of the credential before it expires. This method is used to either extend or reduce the validity period of the credential, but it does *not* reset the password or any other credential attributes.

To reset the validity of the credential:

1. Depending on the type of credential that has to be reset, use the respective **<***CredentialName***>**Input class to obtain an object that implements the class.

   The input required for each credential is different. For example, password is needed for password as well as ArcotID PKI credential, while questions and corresponding answers are required for QnA credentials.

   Note: See "Credential Operations Summary" (see page 69) for the input details required by different credentials.

2. Use the CredentialInput abstract class to obtain the methods that set the common information of the credential.

3. Invoke the CredentialInputList class to pass the input classes of different credentials.

4. (**Optional**) If you are implementing a plug-in, then invoke the setAdditionalInput() method in the AdditionalInput class to fill the AdditionalInput.

   This class provides the additional information that is set as a name-value pair. See "Preparing Additional Input" (see page 45) for more information.

5. Invoke the resetValidity() method of the CredentialIssuance interface to reset the validity of the credential.

   This method returns an instance of the CredentialResponse interface, which specifies the details of all credentials and the transaction.

Handling Errors

Exceptions are returned for any errors that occurred while executing any of the Issuance API methods. The com.arcot.webfort.issuance.api.exception and com.arcot.webfort.common.api.exception packages contain these exceptions. See "Issuance Exceptions" (see page 146) and "Common Exceptions" **(see page 145)** for more information on exception classes.

Note that if no exception is thrown, then processing was successful and the return object can be referenced for processing results. The lack of an error does not necessarily mean that the request was successful.

Resetting Custom Attributes

The custom attributes associated with the credentials can be reset. The CredentialIssuance interface provides resetNotes() method, which helps to reset the custom attributes of the credential.

To reset the custom attributes of the credential:

1. Depending on the type of credential for which the attributes have to be reset, use the respective **<***CredentialName*>Input class to obtain an object that implements the class.

   The input required for each credential is different. For example, password is needed for password as well as ArcotID PKI credential, while questions and corresponding answers are required for QnA credentials.

   Note: See "Credential Operations Summary" (see page 69) for the input details required by different credentials.

2. Use the CredentialInput abstract class to obtain the methods that set the common information of the credential.
3. Invoke the CredentialInputList class to pass the input classes of different credentials.
4. (**Optional**) If you are implementing a plug-in, then invoke the setAdditionalInput() method in the AdditionalInput class to fill the AdditionalInput.

   This class provides the additional information that is set as a name-value pair. See "Preparing Additional Input" (see page 45) for more information.

5. Invoke the resetNotes() method of the CredentialIssuance interface to reset the custom attributes.

   This method returns an instance of the CredentialResponse interface, which specifies the details of all credentials and the transaction.

Handling Errors

Exceptions are returned for any errors that occurred while executing any of the Issuance API methods. The com.arcot.webfort.issuance.api.exception and com.arcot.webfort.common.api.exception packages contain these exceptions. See "Issuance Exceptions" (see page 146) and "Common Exceptions" (see page 145) for more information on exception classes.

Note that if no exception is thrown, then processing was successful and the return object can be referenced for processing results. The lack of an error does not necessarily mean that the request was successful.

Fetching QnA Configuration

The number of questions that the user must set for QnA authentication might vary for every organization. This section explains how to use SDK to fetch this information:

1. (**Optional**) If you are implementing a plug-in, then invoke the setAdditionalInput() method in the AdditionalInput class to fill the AdditionalInput.

   This class provides the additional information that is set as a name-value pair. See "Preparing Additional Input" (see page 45) for more information.

2. Invoke the fetchQnAConfiguration() method of the CredentialIssuance interface to fetch the number of questions.

   This method returns an instance of the CredentialResponse interface, which specifies the details of all credentials and the transaction.

Adding Elements to ArcotID PKI Key Bag

ArcotID PKI can *also* be used to securely store the Open PKI keys and certificates. These keys are typically used for different applications or operations such as, email signing (S/MIME), document signing, and certificate-based authentication (open PKI).

The location where the open PKI keys and certificates are stored in the ArcotID PKI is called *key bag* or *key vault*.

To add elements to ArcotID PKI key bag, you need to implement the addElements() method in the ArcotIDKBMIssuance interface.

1. Use the ArcotIDKeyBagElementSelection and ArcotIDKeyBagElementSet classes to obtain the elements that you need to add to the key bag.
2. (**Optional**) If you are implementing a plug-in, then invoke the setAdditionalInput() method in the AdditionalInput class to fill the AdditionalInput.

   This class provides the additional information that is set as a name-value pair. See "Preparing Additional Input" (see page 45) for more information.
3. Invoke the addElements() method of the ArcotIDKBMIssuance interface to add elements to the ArcotID PKI key bag.

   This method returns an instance of the TransactionDetails interface, which specifies the transaction ID, message, response code, and reason code.

Handling Errors

Exceptions are returned for any errors that occurred while executing any of the Issuance API methods. The com.arcot.webfort.issuance.api.exception and com.arcot.webfort.common.api.exception packages contain these exceptions. See "Issuance Exceptions" (see page 146) and "Common Exceptions" **(see page 145)** for more information on exception classes.

Note that if no exception is thrown, then processing was successful and the return object can be referenced for processing results. The lack of an error does not necessarily mean that the request was successful.

Fetching ArcotID PKI Key Bag Elements

To fetch elements of the ArcotID PKI key bag, you need to implement the getElements() method in the ArcotIDKBMIssuance interface.

1. Use the ArcotIDKeyBagElementSelection and ArcotIDKeyBagElementSet classes to obtain the elements that you need to fetch from the key bag.
2. (**Optional**) If you are implementing a plug-in, then invoke the setAdditionalInput() method in the AdditionalInput class to fill the AdditionalInput.

   This class provides the additional information that is set as a name-value pair. See "Preparing Additional Input" (see page 45) for more information.
3. Invoke the getElements() method of the ArcotIDKBMIssuance interface to fetch elements of the ArcotID PKI key bag.

   This method returns an instance of the TransactionDetails interface, which specifies the transaction ID, message, response code, and reason code.

Handling Errors

Exceptions are returned for any errors that occurred while executing any of the Issuance API methods. The com.arcot.webfort.issuance.api.exception and com.arcot.webfort.common.api.exception packages contain these exceptions. See "Issuance Exceptions" (see page 146) and "Common Exceptions" **(see page 145)** for more information on exception classes.

Note that if no exception is thrown, then processing was successful and the return object can be referenced for processing results. The lack of an error does not necessarily mean that the request was successful.

Deleting ArcotID PKI Key Bag Elements

To delete elements of the ArcotID PKI key bag, you need to implement the deleteElements() method in the ArcotIDKBMIssuance interface.

1. Get the unique identifier of the element that you want to delete from the key bag.
2. (**Optional**) If you are implementing a plug-in, then invoke the setAdditionalInput() method in the AdditionalInput class to fill the AdditionalInput.

   This class provides the additional information that is set as a name-value pair. See "Preparing Additional Input" (see page 45) for more information.
3. Invoke the deleteElements() method of the ArcotIDKBMIssuance interface to delete elements of the ArcotID PKI key bag.

   This method returns an instance of the TransactionDetails interface, which specifies the transaction ID, message, response code, and reason code.

Handling Errors

Exceptions are returned for any errors that occurred while executing any of the Issuance API methods. The com.arcot.webfort.issuance.api.exception and com.arcot.webfort.common.api.exception packages contain these exceptions. See "Issuance Exceptions" (see page 146) and "Common Exceptions" **(see page 145)** for more information on exception classes.

Note that if no exception is thrown, then processing was successful and the return object can be referenced for processing results. The lack of an error does not necessarily mean that the request was successful.

Setting Unsigned Attributes

To set the unsigned attributes for the ArcotID PKI of user, you need to implement the setArcotIDUnsignedAttributes() method.

Note: This operation is applicable *only* for ArcotID PKI credential.

1. Use the ArcotIDAttributes class to set the ArcotID PKI unsigned attributes.
2. (**Optional**) If you are implementing a plug-in, then invoke the setAdditionalInput() method in the AdditionalInput class to fill the AdditionalInput.

   This class provides the additional information that is set as a name-value pair. See "Preparing Additional Input" (see page 45) for more information.
3. Invoke the setArcotIDUnsignedAttributes() method of the CredentialIssuance interface to set the ArcotID PKI unsigned attributes.

   This method returns an instance of the TransactionDetails interface, which specifies the transaction ID, message, response code, and reason code.

Handling Errors

Exceptions are returned for any errors that occurred while executing any of the Issuance API methods. The com.arcot.webfort.issuance.api.exception and com.arcot.webfort.common.api.exception packages contain these exceptions. See "Issuance Exceptions" (see page 146) and "Common Exceptions" **(see page 145)** for more information on exception classes.

Note that if no exception is thrown, then processing was successful and the return object can be referenced for processing results. The lack of an error does not necessarily mean that the request was successful.

Deleting Unsigned Attributes

To delete the unsigned attributes for the ArcotID PKI of a user, you need to implement the deleteArcotIDUnsignedAttributes() method.

Note: This operation is applicable *only* for ArcotID PKI credential.

Perform the following steps to delete the unsigned attributes of the ArcotID PKI:

1. (**Optional**) If you are implementing a plug-in, then invoke the setAdditionalInput() method in the AdditionalInput class to fill the AdditionalInput.

   This class provides the additional information that is set as a name-value pair. See "Preparing Additional Input" (see page 45) for more information.

2. Invoke the deleteArcotIDUnsignedAttributes() method of the CredentialIssuance interface to delete the ArcotID PKI unsigned attributes.

   This method returns an instance of the TransactionDetails interface, which specifies the transaction ID, message, response code, and reason code.

Handling Errors

Exceptions are returned for any errors that occurred while executing any of the Issuance API methods. The com.arcot.webfort.issuance.api.exception and com.arcot.webfort.common.api.exception packages contain these exceptions. See "Issuance Exceptions" (see page 146) and "Common Exceptions" **(see page 145)** for more information on exception classes.

Note that if no exception is thrown, then processing was successful and the return object can be referenced for processing results. The lack of an error does not necessarily mean that the request was successful.

Deleting Credentials

To delete the credentials of a user:

1. Depending on the type of credential you want to delete, use the respective <*CredentialName*>Input class to obtain an object that implements the class.

   The input required for each credential is different. For example, password is needed for password as well as ArcotID PKI credential, while questions and corresponding answers are required for QnA credentials.

   Note: See "Credential Operations Summary" (see page 69) for the input details required by different credentials.

2. Use the CredentialInput abstract class to obtain the methods that set the common information of the credential.

3. Invoke the CredentialInputList class to pass the input classes of different credentials.

4. (**Optional**) If you are implementing a plug-in, then invoke the setAdditionalInput() method in the AdditionalInput class to fill the AdditionalInput.

   This class provides the additional information that is set as a name-value pair. See "Preparing Additional Input" (see page 45) for more information.

5. Invoke the delete() method of the CredentialIssuance interface to delete the credential.

   This method returns an instance of the ConfigurationResponse interface, which specifies the QnA configuration details.

Handling Errors

Exceptions are returned for any errors that occurred while executing any of the Issuance API methods. The com.arcot.webfort.issuance.api.exception and com.arcot.webfort.common.api.exception packages contain these exceptions. See "Issuance Exceptions" (see page 146) and "Common Exceptions" **(see page 145)** for more information on exception classes.

Note that if no exception is thrown, then processing was successful and the return object can be referenced for processing results. The lack of an error does not necessarily mean that the request was successful.

Reading the Output

The following table lists the methods that fetch the credential and user details:

Note: Most of the methods listed in the following table can also return NULL.

Method	**Description**
Common Output Methods	
getCreateTime()	Fetches the time when the credential was created.
getDisableEndTime()	Fetches the time when the disabled credential has to be enabled.
getDisableStartTime()	Fetches the time when the credential has to be disabled.
getLastFailedAuthAttemptTime()	Fetches the time when the last authentication attempt failed.
getLastSuccessAuthAttemptTime()	Fetches the time when the last authentication attempt succeeded.
getLastUpdatedTime()	Fetches the time when the credential was updated last time.
getNotes()	Fetches the custom attributes that are set for the credential.
getNumberOfFailedAuthAttempts()	Fetches the total number of failed authentication attempts for the user.
getOrgName()	Fetches the organization name to which the user belongs.
getProfileName()	Fetches the profile name with which the credential was created.
getProfileVersion()	Fetches the version number of the profile.
getRemainingUsageCount()	Fetches the number of times the credential can be used.
getStatus()	Fetches the status of the credential.
getUserName()	Fetches the name of the authenticating user.
getValidityEndTime()	Fetches the date after which the credential expires.
getValidityStartTime()	Fetches the date from when the credential is valid.
ArcotID Output Method	
getUnsignedAttributes()	Fetches the unsigned attributes of the ArcotID PKI that the user has set.

Method	Description
QnA Output Method	
getQuestions()	Fetches the questions set for the user.
Password Output Method	
Only the common output methods are available for this authentication method.	
One-Time Password Output Methods	
getOTP()	Fetches the One-Time Password (OTP) for the user.
getUsageCount()	Fetches the number of times the OTP can be used.
OATH One-Time Password Output Methods	
getCounterOffSet()	Fetches the OATH OTP count on the server.
getLength()	Fetches the length the OATH OTP issued to the user.
getTokenID()	Fetches the OATH OTP token ID.
getType()	Fetches the type of OATH OTP (HOTP or TOTP) that has to be issued to the user.
ArcotOTP Output Methods	
getCard()	Fetches the ArcotID OTP card generated by the AuthMinder Server.
getCounterOffSet()	Fetches the ArcotID OTP count on the server.
getType()	Fetches the type of ArcotID OTP (HOTP or TOTP).
EMV OTP Output Methods	
getCard()	Fetches the OTP card generated by the AuthMinder Server.
getCounterOffSet()	Fetches the EMV OTP count on the server.

Credential Operations Summary

This section provides the input parameters required for performing lifecycle management operations for each credential and the expected output for:

- ArcotID PKI Operations (see page 69)
- Password Operations (see page 73)
- Question and Answer Operations (see page 76)
- One-Time Password Operations (see page 79)
- OATH OTP Operations (see page 81)
- ArcotID OTP Operations (see page 83)
- EMV OTP Operations (see page 86)

ArcotID PKI Operations

The following table provides the input and output information for ArcotID PKI operations:

Operation (Function Used)	**Input Required**	**Expected Output**
Creating credential (create())	■ User name (userName). ■ (Optional) Organization name (orgName). ■ ArcotID password (password). ■ (Optional) ArcotID attributes (unsignedAttributes). ■ (Optional) Additional Input (AdditionalInput). ■ (Optional) Profile name of the credential. If it is not passed, then the default profile for the organization is used. If it is passed, then the profile name must be available at the organization. ■ (Optional) Custom attributes that you have maintained for each credential in your application.	■ CredentialDetails ■ TransactionDetails

Operation (Function Used)	**Input Required**	**Expected Output**
Resetting credential (resetCredentail())	■ User name (userName). ■ (Optional) Organization name (orgName). ■ ArcotID password (password). ■ (Optional) ArcotID attributes (unsignedAttributes). ■ (Optional) Additional Input (AdditionalInput). ■ (Optional) Profile name of the credential.	■ CredentialDetails ■ TransactionDetails
Fetching credential (fetch())	■ User name (userName). ■ (Optional) Organization name (orgName). ■ (Optional) Additional Input (AdditionalInput). ■ (Optional) Profile name of the credential.	■ CredentialDetails ■ TransactionDetails
Reissuing credential (reissue())	■ User name (userName). ■ (Optional) Organization name (orgName). ■ ArcotID password (password). ■ (Optional) ArcotID attributes (unsignedAttributes). ■ (Optional) Additional Input (AdditionalInput). ■ (Optional) Profile name of the credential.	■ CredentialDetails ■ TransactionDetails

Operation (Function Used)	**Input Required**	**Expected Output**
Resetting credential validity (resetValidity())	■ User name (userName). ■ (Optional) Organization name (orgName). ■ (Optional) Additional Input (AdditionalInput). ■ Time when the validity of the credential ends (validityEndTimeEx). ■ (Optional) Profile name of the credential.	■ CredentialDetails ■ TransactionDetails
Resetting custom attributes (resetNotes())	■ User name (userName). ■ (Optional) Organization name (orgName). ■ (Optional) Additional Input (AdditionalInput). ■ Custom attributes that you have maintained in your application. ■ (Optional) Profile name of the credential.	■ CredentialDetails ■ TransactionDetails
Disabling credential (disable()) Enabling credential (enable()) Deleting credential (delete())	■ User name (userName). ■ (Optional) Organization name (orgName). ■ (Optional) Additional Input (AdditionalInput). ■ (Optional) Profile name of the credential.	■ CredentialDetails ■ TransactionDetails
Setting ArcotID unsigned attribute (setArcotIDUnsignedAttributes())	■ User name (userName). ■ (Optional) Organization name (orgName). ■ (Optional) Additional Input (AdditionalInput). ■ ArcotID attributes (unsignedAttributes). ■ (Optional) Profile name of the credential.	TransactionDetails

Operation (Function Used)	Input Required	Expected Output
Adding key bag attributes (addElements())	■ User name (userName). ■ (Optional) Organization name (orgName). ■ (Optional) Additional Input (AdditionalInput). ■ Certificate elements (elementSet). ■ ArcotID key bag details (ArcotIDKeyBagElementSlection). ■ (Optional) Profile name of the credential.	TransactionDetails
Fetching key bag attributes (getElements())	■ User name (userName). ■ (Optional) Organization name (orgName). ■ (Optional) Additional Input (AdditionalInput). ■ ArcotID key bag details (ArcotIDKeyBagElementSlection). ■ (Optional) Profile name of the credential.	
Deleting key bag attributes (deleteElements())	■ User name (userName). ■ (Optional) Organization name (orgName). ■ (Optional) Additional Input (AdditionalInput). ■ Certificate element identifiers (elementIds). ■ (Optional) Profile name of the credential.	
Downloading Credential (downloadCredential())	■ User name (userName). ■ (Optional) Organization name (orgName). ■ (Optional) Additional Input (AdditionalInput). ■ (Optional) Profile name of the credential.	

Operation (Function Used)	Input Required	Expected Output
Deleting unsigned attribute (deleteArcotIDUnsignedAttributes())	■ User name (userName). ■ (Optional) Organization name (orgName). ■ (Optional) Additional Input (AdditionalInput). ■ Array of ArcotID unsigned attributes. ■ (Optional) Profile name of the credential.	

Password Operations

The following table provides the input and output information for password operations:

Operation (Function Used)	Input Required	Expected Output
Creating credential (create())	■ User name (userName). ■ (Optional) Organization name (orgName). ■ Password (password). ■ (Optional) Additional Input (AdditionalInput). ■ (Optional) Profile name of the credential. If it is not passed, then the default profile for the organization is used. If it is passed, then the profile name must be available at the organization. ■ (Optional) Custom attributes that you have maintained for each credential in your application.	■ CredentialDetails ■ TransactionDetails

Operation (Function Used)	**Input Required**	**Expected Output**
Resetting credential (resetCredentail())	■ User name (userName). ■ (Optional) Organization name (orgName). ■ Password (password). ■ (Optional) Additional Input (AdditionalInput). ■ (Optional) Profile name of the credential.	■ CredentialDetails ■ TransactionDetails
Fetching credential (fetch())	■ User name (userName). ■ (Optional) Organization name (orgName). ■ (Optional) Additional Input (AdditionalInput). ■ (Optional) Profile name of the credential.	■ CredentialDetails ■ TransactionDetails
Reissuing credential (reissue())	■ User name (userName). ■ (Optional) Organization name (orgName). ■ Password (password). ■ (Optional) Additional Input (AdditionalInput). ■ (Optional) Profile name of the credential.	■ CredentialDetails ■ TransactionDetails
Resetting credential validity (resetValidity())	■ User name (userName). ■ (Optional) Organization name (orgName). ■ (Optional) Additional Input (AdditionalInput). ■ Time when the validity of the credential ends (validityEndTimeEx). ■ (Optional) Profile name of the credential.	■ CredentialDetails ■ TransactionDetails

Operation (Function Used)	**Input Required**	**Expected Output**
Resetting custom attributes (resetNotes())	■ User name (userName). ■ (Optional) Organization name (orgName). ■ (Optional) Additional Input (AdditionalInput). ■ Custom attributes that you have maintained in your application. ■ (Optional) Profile name of the credential.	■ CredentialDetails ■ TransactionDetails
Disabling credential (disable())	■ User name (userName). ■ (Optional) Organization name (orgName). ■ (Optional) Additional Input (AdditionalInput). ■ (Optional) Profile name of the credential.	■ CredentialDetails ■ TransactionDetails
Enabling credential (enable())		
Deleting credential (delete())		

Question and Answer Operations

The following table provides the input and output information for QnA operations:

Operation (Function Used)	Input Required	Expected Output
Creating credential (create())	■ User name (userName). ■ (Optional) Organization name (orgName). ■ List of questions and answers (question and answer). ■ (Optional) Additional Input (AdditionalInput). ■ (Optional) Profile name of the credential. If it is not passed, then the default profile for the organization is used. If it is passed, then the profile name must be available at the organization. ■ (Optional) Custom attributes that you have maintained for each credential in your application.	■ CredentialDetails ■ TransactionDetails
Resetting credential (resetCredentail())	■ User name (userName). ■ (Optional) Organization name (orgName). ■ List of questions and answers (question and answer). ■ (Optional) Additional Input (AdditionalInput). ■ (Optional) Profile name of the credential. If it is not passed, then the default profile for the organization is used. If it is passed, then the profile name must be available at the organization.	■ CredentialDetails ■ TransactionDetails

Operation (Function Used)	**Input Required**	**Expected Output**
Fetching credential (fetch())	■ User name (userName). ■ (Optional) Organization name (orgName). ■ (Optional) Additional Input (AdditionalInput). ■ (Optional) Profile name of the credential.	■ CredentialDetails ■ TransactionDetails
Fetching credential configuration (fetchQnAConfiguration())	■ (Optional) Organization name (orgName). ■ (Optional) Profile name of the credential. If it is not passed, then the default profile for the organization is used. If it is passed, then the profile name must be available at the organization. ■ (Optional) Additional Input (AdditionalInput).	■ QnAConfiguration ■ TransactionDetails
Fetching credential configuration (fetchQnAConfiguration())	■ (Optional) Organization name (orgName). ■ (Optional) Profile name of the credential. If it is not passed, then the default profile for the organization is used. If it is passed, then the profile name must be available at the organization. ■ (Optional) Additional Input (AdditionalInput). ■ Questions to fetch (fetchQuestions)	■ QnAConfiguration ■ TransactionDetails ■ questions
Reissuing credential (reissue())	■ User name (userName). ■ (Optional) Organization name (orgName). ■ List of questions and answers (question and answer). ■ (Optional) Additional Input (AdditionalInput). ■ (Optional) Profile name of the credential.	■ CredentialDetails ■ TransactionDetails

Operation (Function Used)	**Input Required**	**Expected Output**
Resetting credential validity (resetValidity())	■ User name (userName). ■ (Optional) Organization name (orgName). ■ (Optional) Additional Input (AdditionalInput). ■ Time when the validity of the credential ends (validityEndTimeEx). ■ (Optional) Profile name of the credential.	■ CredentialDetails ■ TransactionDetails
Resetting custom attributes (resetNotes())	■ User name (userName). ■ (Optional) Organization name (orgName). ■ (Optional) Additional Input (AdditionalInput). ■ (Optional) Profile name of the credential.	■ CredentialDetails ■ TransactionDetails
Disabling credential (disable())	■ User name (userName). ■ (Optional) Organization name (orgName). ■ (Optional) Additional Input (AdditionalInput). ■ (Optional) Profile name of the credential.	■ CredentialDetails ■ TransactionDetails
Enabling credential (enable())		
Deleting credential (delete())		

One-Time Password Operations

The following table provides the input and output information for OTP operations:

Operation (Function Used)	Input Required	Expected Output
Creating credential (create())	■ User name (userName). ■ (Optional) Organization name (orgName). ■ (Optional) Additional Input (AdditionalInput). ■ (Optional) Profile name of the credential. If it is not passed, then the default profile for the organization is used. If it is passed, then the profile name must be available at the organization. ■ (Optional) Custom attributes that you have maintained for each credential in your application.	■ CredentialDetails ■ TransactionDetails ■ Password
Fetching credential (fetch())	■ User name (userName). ■ (Optional) Organization name (orgName). ■ (Optional) Additional Input (AdditionalInput). ■ (Optional) Profile name of the credential.	■ CredentialDetails ■ TransactionDetails
Resetting credential (resetCredentail())	■ User name (userName). ■ (Optional) Organization name (orgName). ■ (Optional) Additional Input (AdditionalInput). ■ (Optional) Profile name of the credential.	■ CredentialDetails ■ TransactionDetails ■ Password

Operation (Function Used)	**Input Required**	**Expected Output**
Resetting credential validity (resetValidity())	■ User name (userName). ■ (Optional) Organization name (orgName). ■ (Optional) Additional Input (AdditionalInput). ■ Time when the validity of the credential ends (validityEndTimeEx). ■ (Optional) Profile name of the credential.	■ CredentialDetails ■ TransactionDetails
Reissuing credential (reissue())	■ User name (userName). ■ (Optional) Organization name (orgName). ■ (Optional) Additional Input (AdditionalInput). ■ (Optional) Profile name of the credential.	■ CredentialDetails ■ TransactionDetails
Resetting custom attributes (resetNotes())	■ User name (userName). ■ (Optional) Organization name (orgName). ■ (Optional) Additional Input (AdditionalInput). ■ Custom attributes that you have maintained in your application. ■ (Optional) Profile name of the credential.	■ CredentialDetails ■ TransactionDetails
Disabling credential (disable())	■ User name (userName). ■ (Optional) Organization name (orgName). ■ (Optional) Additional Input (AdditionalInput). ■ (Optional) Profile name of the credential.	■ CredentialDetails ■ TransactionDetails
Enabling credential (enable())		
Deleting credential (delete())		

OATH OTP Operations

The following table provides the input and output information for OATH OTP operations:

Operation (Function Used)	**Input Required**	**Expected Output**
Creating credential (create())	■ User name (userName). ■ (Optional) Organization name (orgName). ■ (Optional) Additional Input (AdditionalInput). ■ (Optional) Profile name of the credential. If it is not passed, then the default profile for the organization is used. If it is passed, then the profile name must be available at the organization. ■ (Optional) Custom attributes that you have maintained for each credential in your application.	■ CredentialDetails ■ TransactionDetails ■ Password
Fetching credential (fetch())	■ User name (userName). ■ (Optional) Organization name (orgName). ■ (Optional) Additional Input (AdditionalInput). ■ (Optional) Profile name of the credential.	■ CredentialDetails ■ TransactionDetails
Resetting credential (resetCredentail())	■ User name (userName). ■ (Optional) Organization name (orgName). ■ (Optional) Additional Input (AdditionalInput). ■ (Optional) Profile name of the credential.	■ CredentialDetails ■ TransactionDetails ■ Password

Operation (Function Used)	**Input Required**	**Expected Output**
Resetting credential validity (resetValidity())	■ User name (userName). ■ (Optional) Organization name (orgName). ■ (Optional) Additional Input (AdditionalInput). ■ Time when the validity of the credential ends (validityEndTimeEx). ■ (Optional) Profile name of the credential.	■ CredentialDetails ■ TransactionDetails
Reissuing credential (reissue())	■ User name (userName). ■ (Optional) Organization name (orgName). ■ (Optional) Additional Input (AdditionalInput). ■ (Optional) Profile name of the credential.	■ CredentialDetails ■ TransactionDetails
Downloading Credential (downloadCredential())	■ User name (userName). ■ (Optional) Organization name (orgName). ■ (Optional) Additional Input (AdditionalInput). ■ (Optional) Profile name of the credential. ■ Password (password).	■ CredentialDetails ■ TransactionDetails
Resetting custom attributes (resetNotes())	■ User name (userName). ■ (Optional) Organization name (orgName). ■ (Optional) Additional Input (AdditionalInput). ■ Custom attributes that you have maintained in your application. ■ (Optional) Profile name of the credential.	■ CredentialDetails ■ TransactionDetails

Operation (Function Used)	**Input Required**	**Expected Output**
Disabling credential (disable())	■ User name (userName). ■ (Optional) Organization name (orgName). ■ (Optional) Additional Input (AdditionalInput). ■ (Optional) Profile name of the credential.	■ CredentialDetails ■ TransactionDetails
Enabling credential (enable())		
Deleting credential (delete())		

ArcotID OTP Operations

The following table provides the input and output information for ArcotID OTP operations:

Operation (Function Used)	**Input Required**	**Expected Output**
Creating credential (create())	■ User name (userName). ■ (Optional) Organization name (orgName). ■ Password (password) ■ (Optional) Additional Input (AdditionalInput). ■ (Optional) Profile name of the credential. If it is not passed, then the default profile for the organization is used. If it is passed, then the profile name must be available at the organization. ■ (Optional) Custom attributes that you have maintained for each credential in your application.	■ CredentialDetails ■ TransactionDetails ■ Password

Operation (Function Used)	**Input Required**	**Expected Output**
Fetching credential (fetch())	■ User name (userName). ■ (Optional) Organization name (orgName). ■ (Optional) Additional Input (AdditionalInput). ■ (Optional) Profile name of the credential.	■ CredentialDetails ■ TransactionDetails
Resetting credential (resetCredentail())	■ User name (userName). ■ (Optional) Organization name (orgName). ■ Password (password) ■ (Optional) Additional Input (AdditionalInput). ■ (Optional) Profile name of the credential.	■ CredentialDetails ■ TransactionDetails ■ Password
Resetting credential validity (resetValidity())	■ User name (userName). ■ (Optional) Organization name (orgName). ■ (Optional) Additional Input (AdditionalInput). ■ Time when the validity of the credential ends (validityEndTimeEx). ■ (Optional) Profile name of the credential.	■ CredentialDetails ■ TransactionDetails
Reissue (reset())	■ User name (userName). ■ (Optional) Organization name (orgName). ■ Password (password) ■ (Optional) Additional Input (AdditionalInput). ■ (Optional) Profile name of the credential.	■ CredentialDetails ■ TransactionDetails ■ Password

Operation (Function Used)	**Input Required**	**Expected Output**
Resetting custom attributes (resetNotes())	■ User name (userName). ■ (Optional) Organization name (orgName). ■ (Optional) Additional Input (AdditionalInput). ■ Custom attributes that you have maintained in your application. ■ (Optional) Profile name of the credential.	■ CredentialDetails ■ TransactionDetails
Downloading Credential (downloadCredential())	■ User name (userName). ■ (Optional) Organization name (orgName). ■ (Optional) Additional Input (AdditionalInput). ■ (Optional) Profile name of the credential.	■ CredentialDetails ■ TransactionDetails
Disabling credential (disable())	■ User name (userName). ■ (Optional) Organization name (orgName). ■ (Optional) Additional Input (AdditionalInput). ■ (Optional) Profile name of the credential.	■ CredentialDetails ■ TransactionDetails
Enabling credential (enable())		
Deleting credential (delete())		

EMV OTP Operations

The following table provides the input and output information for EMV OTP operations:

Operation (Function Used)	**Input Required**	**Expected Output**
Creating credential (create())	■ User name (userName). ■ (Optional) Organization name (orgName). ■ (Optional) Additional Input (AdditionalInput). ■ (Optional) Profile name of the credential. If it is not passed, then the default profile for the organization is used. If it is passed, then the profile name must be available at the organization. ■ (Optional) Custom attributes that you have maintained for each credential in your application.	■ CredentialDetails ■ TransactionDetails ■ Password
Fetching credential (fetch())	■ User name (userName). ■ (Optional) Organization name (orgName). ■ (Optional) Additional Input (AdditionalInput). ■ (Optional) Profile name of the credential.	■ CredentialDetails ■ TransactionDetails
Resetting credential (resetCredentail())	■ User name (userName). ■ (Optional) Organization name (orgName). ■ (Optional) Additional Input (AdditionalInput). ■ (Optional) Profile name of the credential.	■ CredentialDetails ■ TransactionDetails ■ Password

Operation (Function Used)	**Input Required**	**Expected Output**
Resetting credential validity (resetValidity())	■ User name (userName). ■ (Optional) Organization name (orgName). ■ (Optional) Additional Input (AdditionalInput). ■ Time when the validity of the credential ends (validityEndTimeEx). ■ (Optional) Profile name of the credential.	■ CredentialDetails ■ TransactionDetails
Reissue (reissue())	■ User name (userName). ■ (Optional) Organization name (orgName). ■ (Optional) Additional Input (AdditionalInput). ■ (Optional) Profile name of the credential.	■ CredentialDetails ■ TransactionDetails ■ Password
Resetting custom attributes (resetNotes())	■ User name (userName). ■ (Optional) Organization name (orgName). ■ (Optional) Additional Input (AdditionalInput). ■ Custom attributes that you have maintained in your application. ■ (Optional) Profile name of the credential.	■ CredentialDetails ■ TransactionDetails
Downloading Credential (downloadCredential())	■ User name (userName). ■ (Optional) Organization name (orgName). ■ (Optional) Additional Input (AdditionalInput). ■ (Optional) Profile name of the credential. ■ Password (password).	■ CredentialDetails ■ TransactionDetails

Operation (Function Used)	**Input Required**	**Expected Output**
Disabling credential (disable())	■ User name (userName). ■ (Optional) Organization name (orgName). ■ (Optional) Additional Input (AdditionalInput). ■ (Optional) Profile name of the credential.	■ CredentialDetails ■ TransactionDetails
Enabling credential (enable())		
Deleting credential (delete())		

Chapter 7: Authenticating Users

This chapter introduces you to the APIs that are used for different authentication methods supported by AuthMinder. This chapter covers the following topics:

Initializing the Authentication SDK

Initialize the Authentication SDK by using the Authentication class in the com.arcot.webfort.authentication.api package. The initialization process caches all the database tables, creates the database pool, and loggers. After initialization, it returns an appropriate message to the calling application.

Note: You *cannot* apply any configuration changes after you initialize the API. To enable the configuration changes, you must re-initialize the API.

The Authentication class provides two methods to initialize the Authentication SDK.

Method 1: Initializing the SDK by Using the Map

This method initializes the Authentication SDK based on the map provided. The following table provides the details of the init() method:

Description	**Input Values**	**Output Value**
Initializes the Authentication SDK by using the provided map.	■ map The key-value pair specifying the configuration information. The map values are same as the parameters listed in the webfort.authentication.properties file. **Book:** Refer to appendix, "Configuration Files and Options" in the *CA AuthMinder Installation and Deployment Guide* for more information on the parameters in this properties file. ■ locale The locale of the API. The default value is set to en_US.	Returns an exception if the SDK is not initialized successfully.

Method 2: Initializing the SDK by Using the Properties File

This method initializes the Authentication SDK by using the parameters listed in the properties file. If you pass NULL, then the parameters are read from the webfort.authentication.properties file. If you provide a different file name containing these configuration parameters, then that file is read instead.

The following table provides the details of the initialization method using the properties method:

Description	**Input Values**	**Output Value**
Initializes the Authentication SDK by using the properties file.	■ location The absolute path of the properties file. Be default, this file is available at *<install_location>*/sdk/client/java/properties ■ locale The locale of the API. The default value is set to en_US.	Returns an exception if the SDK is not initialized successfully.

Releasing the Authentication API Resources

The Authentication class also provides a method to release the resources such as sockets that are used by Authentication SDK.

Important! This method must be invoked before re-initializing the SDK.

The following table provides the details of the release() method:

Description	Input Values	Output Value
Releases the Authentication SDK.	The locale of the API.	Returns an exception if the API is not released successfully.

Preparing Additional Input

You need to prepare additional inputs if you plan to augment AuthMinder's standard authentication capability by implementing plug-ins, then you need to set the extra information that must be sent to AuthMinder Server in name-value pairs. AuthMinder's com.arcot.webfort.common.api provides you the AdditionalInput class, which enables you to set this additional information that you plan to use.

Some of the pre-defined additional input parameters supported by the AdditionalInput class are:

- AR_WF_LOCALE_ID

  Specifies the locale that AuthMinder will use in returning the messages back to the calling application.

- AR_WF_CALLER_ID

  This is useful in tracking transactions. You can use session ID or transaction ID for specifying this information.

- AR_WF_TXN_FILE_LOG_TRACE

  Enables the TRACE logging for the transaction. The presence of the identifier irrespective of the value enables TRACE logging.

- AR_WF_TXN_FILE_LOG_LEVEL

  Used to control the log level for the transaction. The supported values are:

  - 1 for WARNING
  - 2 for INFO
  - 3 for DETAIL

  See appendix, "AuthMinder Logging" (see page 125) for more information on log levels.

- AR_WF_TXN_LOG_SENSITIVE_DATA

  Is used to indicate whether the sensitive data must be logged for the current transaction. For example, USERNAME of the user. The presence of the identifier irrespective of the value enables this logging.

- AR_WF_TXN_DB_LOG_QUERY_DETAILS

  Is used to indicate whether the database query execution has to be logged in detail. The presence of the identifier irrespective of the value enables this logging.

- AR_WF_OTP_TXN_SIGN_DATA

  Specifies the transaction data that the end user enters in the Challenge field of the ArcotID OTP client to generate a passcode in the Sign mode. The maximum length of the signed data is 64 bytes. This implementation of the Transaction Signing feature conforms to the OATH Challenge-Response Algorithm (OCRA) as defined by RFC 6287.

ArcotID PKI Authentication

ArcotID PKI is a challenge-response type of authentication, where AuthMinder Server provides a challenge. The signed challenge is sent by the ArcotID PKI Client to the AuthMinder Server through the application. The following topics are explained in this section:

1. ArcotID PKI Download
2. ArcotID PKI Authentication

For successful ArcotID PKI authentication, you must ensure that you have integrated ArcotID PKI Client with application, as discussed in chapter, "Integrating ArcotID PKI Client with Application" (see page 33).

Note: The ArcotID PKI download and authentication can be in multiple ways, see chapter, "Understanding AuthMinder WorkFlows" (see page 13) for more information. This section focuses on the APIs that are used for these operations.

ArcotID PKI Download

To perform ArcotID PKI authentication, the ArcotID PKI of the user has to be present on the system from where the authentication request is originating. If the ArcotID PKI is not present, then it needs to be downloaded to the system. In such a case the user must perform a secondary authentication before the ArcotID PKI is downloaded.

To download the ArcotID PKI:

1. (**Optional**) If you are implementing a plug-in, then invoke the setAdditionalInput() method in the AdditionalInput class to obtain an object that implements the class.

   See "Preparing Additional Input" (see page 92) for more information.

2. Invoke the getArcotID() method of the ArcotIDAuth interface to fetch the ArcotID PKI of the user to your application.

   This method returns an instance of the ArcotIDResponse interface, which will have the ArcotID PKI of the user.

3. The user’s ArcotID PKI is set in the HTML or Java Server Page (JSP).
4. Invoke the ImportArcotID() client-side API to download the ArcotID PKI from your application to the end user’s system.

ArcotID PKI Authentication

To perform ArcotID PKI authentication:

1. (**Optional**) If you are implementing a plug-in, then invoke the setAdditionalInput() method in the AdditionalInput class to obtain an object that implements the class.

   See "Preparing Additional Input" (see page 92) for more information.

2. Invoke the getChallenge() method of the ArcotIDAuth interface to retrieve the challenge form the AuthMinder Server.

   This method returns an instance of the ArcotIDChallengeResponse, which has the transaction details and also the challenge from the server.

3. The challenge is sent to the end user through HTML Page.

4. Invoke the ArcotID PKI Client-side method, SignChallengeEx() to sign the challenge.

   The application collects the ArcotID PKI password and the challenge is signed by the ArcotID PKI Client using the ArcotID PKI password.

5. Invoke the verifySignedChallenge() method of the ArcotIDAuth interface to verify the signed challenge. Optionally, you can also specify the token type that must be returned to the user *after* successful authentication by using the AuthTokenType class.

   This method returns an instance of the AuthResponse interface, which provides the transaction details, credential details, and token information.

Questions and Answers Authentication

The *Question and Answer* (*QnA*) authentication mechanism can either be used as a secondary authentication method for ArcotID PKI Download**,** or Forgot Your Password (FYP) authentication, or can be used as an independent authentication type.

In this mechanism, the user can either set their own set of questions and answers during the QnA creation stage, or your application can choose to ask pre-defined questions to the user. The maximum number of questions to be set, the number of questions to be asked to the user, and the minimum correct answers to be collected during authentication are all configurable parameters and can be set by using the Administration Console.

AuthMinder provides a facility called *caller verification*, which enables you to collect the answers from the user, verify the answers, and then send the verification result to the AuthMinder Server.

QnA Authentication Using Caller Verification Feature

The following steps explains how to perform QnA authentication if caller verification feature is enabled:

1. Invoke the getQuestions() method of the QnAAuth interface to retrieve the user’s questions and answers from the AuthMinder Server.

   Note: The QnAAuth interface provides two getQuestions() methods, you must call the method that takes the boolean input (fetchAnswers) to fetch the answers.

   This method returns an instance of the QnAResponse interface, which includes the questions to be asked, answers for each question, transaction ID, message, response code, and reason code.

2. Prepare an object to hold the questions and answers of the user. For this, you must invoke the methods of AuthQnAInfo interface in the following order:

   a. getNumberofQuestions

      Invoke this method to know the number of questions that are set for the user.

   b. getQuestion

      Invoke this method to get the questions that are set for the user. The number of questions fetched by this method depends on the number returned by the getNumberofQuestions() method.

   c. Implement the logic to collect the answers from the user for the questions retrieved from AuthMinder Server.

   d. answerQuestion

      Note: The AuthQnAInfo interface provides two answerQuestion() methods, you must call the method that takes the verification status as one of the input.

      Invoke this method to set the answer collected by the application.

3. (**Optional**) If you are implementing a plug-in, then invoke the setAdditionalInput() method in the AdditionalInput class to obtain an object that implements the class.

   See "Preparing Additional Input" (see page 92) for more information.

4. Invoke the verifyAnswers() method of the QnAAuth interface by passing the AuthQnAInfo object created in Step 2 to verify the answers provided by the user. Optionally, you can also specify the token type that must be returned to the user *after* successful authentication by using the AuthTokenType class.

   This method returns an instance of the AuthResponse interface, which provides the transaction details, credential details, and token information.

QnA Authentication Using Server Verification

The following steps explain how to perform QnA authentication if the QnA credential is directly verified by AuthMinder Server:

1. Invoke the getQuestions() method of the QnAAuth interface to retrieve the user's questions and answers from the AuthMinder Server.

   Note: The QnAAuth interface provides two getQuestions() methods, you must call the method that fetches the questions *only*.

   This method returns an instance of the QnAResponse interface, which includes the questions to be asked, answers for each question, transaction ID, message, response code, and reason code.

2. Prepare an object to hold the questions and answers of the user. For this, you must invoke the methods of AuthQnAInfo interface in the following order:

   a. getNumberofQuestions

      Invoke this method to know the number of questions that are set for the user.

   b. getQuestion

      Invoke this method to get the questions that are set for the user. The number of questions fetched by this method depends on the number returned by the getNumberofQuestions() method.

   c. Implement the logic to collect the answers from the user for the questions retrieved from AuthMinder Server.

   d. answerQuestion

      Note: The AuthQnAInfo interface provides two answerQuestion() methods, you must call the method that takes *does* not take verification status as one of the input.

      Invoke this method to set the answer collected by the application.

3. (**Optional**) If you are implementing a plug-in, then invoke the setAdditionalInput() method in the AdditionalInput class to obtain an object that implements the class.

   See "Preparing Additional Input" (see page 92) for more information.

4. Invoke the verifyAnswers() method of the QnAAuth interface by passing the AuthQnAInfo object created in Step 2 to verify the answers provided by the user. Optionally, you can also specify the token type that must be returned to the user *after* successful authentication by using the AuthTokenType class.

   This method returns an instance of the AuthResponse interface, which provides the transaction details, credential details, and token information.

Password Authentication

The authentication API provides the PasswordAuth interface to perform the traditional *password* authentication. In this authentication mechanism, the user specifies the user name and the corresponding password for authentication. The password entered by the user is then verified.

AuthMinder supports partial password authentication, if you enable this feature, then the user will be challenged to enter the characters in different positions of the password. For example, if the password is *casablanca!*, then the user can be asked to enter the characters in positions 1, 3, and 8, which would be *csn*.

Complete Password Authentication

To perform regular password authentication:

1. Implement the logic to collect the user's password.
2. (**Optional**) If you are implementing a plug-in, then invoke the setAdditionalInput() method in the AdditionalInput class to obtain an object that implements the class.

   See "Preparing Additional Input" (see page 92) for more information.
3. Invoke the verifyPassword() method of the PasswordAuth interface to verify the password provided by the user. Optionally, you can also specify the token type that must be returned to the user *after* successful authentication by using the AuthTokenType class.

   Note: You need to invoke the verifyPassword() method that does not take the challenge identifier (challengeID) as one of the input parameter.

   This method returns an instance of the AuthResponse interface, which provides the transaction details, credential details, and token information.

Partial Password Authentication

To perform partial password authentication:

1. (**Optional**) If you are implementing a plug-in, then invoke the setAdditionalInput() method in the AdditionalInput class to obtain an object that implements the class.

   See "Preparing Additional Input" (see page 92) for more information.

2. Invoke the getPasswordChallenge() method of the PasswordAuth interface to obtain the challenge from the AuthMinder Server.

   This method returns the unique identifier for the challenge and the password character positions that the user has to answer.

3. Implement the logic to collect the user’s password.

4. Invoke the verifyPassword() method of the PasswordAuth interface to verify the password provided by the user. Optionally, you can also specify the token type that must be returned to the user *after* successful authentication by using the AuthTokenType class.

   Note: You need to invoke the verifyPassword() method that takes the challenge identifier (challengeID) as one of the input parameter.

   This method returns an instance of the AuthResponse interface, which provides the transaction details, credential details, and token information.

One-Time Password Authentication

One-Time Password (*OTP*) is a numeric or an alpha-numeric string that is generated by the AuthMinder Server. AuthMinder supports OTPs that can be reused pre-configured number of times. You can specify this setting by using the Administration Console. The OTP lifetime depends on the duration for which it is valid and number of times it can be used.

To perform OTP authentication:

1. Implement the logic to collect the OTP from the user.

2. (**Optional**) If you are implementing a plug-in, then invoke the setAdditionalInput() method in the AdditionalInput class to obtain an object that implements the class.

   See "Preparing Additional Input" (see page 92) for more information.

3. Invoke the verifyOTP() method of the OTPAuth interface to verify the OTP of the user. Optionally, you can also specify the token type that must be returned to the user *after* successful authentication by using the AuthTokenType class.

   This method returns an instance of the AuthResponse interface, which provides the transaction details, credential details, and token information.

OATH One-Time Password Authentication

To authenticate the OTPs that are OATH compliant:

1. Implement the logic to collect the OATH OTP from the user.
2. (**Optional**) If you are implementing a plug-in, then invoke the setAdditionalInput() method in the AdditionalInput class to obtain an object that implements the class.

   See "Preparing Additional Input" (see page 92) for more information.
3. Invoke the verifyOTP() method of the OATHAuth interface to verify the OTP of the user. Optionally, you can also specify the token type that must be returned to the user *after* successful authentication by using the AuthTokenType class.

   This method returns an instance of the AuthResponse interface, which provides the transaction details, credential details, and token information.

OATH One-Time Password Synchronization

To synchronize the clint and server OATH OTP:

1. (**Optional**) If you are implementing a plug-in, then invoke the setAdditionalInput() method in the AdditionalInput class to obtain an object that implements the class.

   See "Preparing Additional Input" (see page 92) for more information.
2. Invoke the synchronizeOTP() method of the OATHAuth interface to synchronize the server OTP with the client OTP. Optionally, you can also specify the token type that must be returned to the user *after* successful authentication by using the AuthTokenType class.

   This method returns an instance of the AuthResponse interface, which provides the transaction details, credential details, and token information.

ArcotID OTP (ArcotID OTP-OATH) Authentication

To authenticate OATH-complaint ArcotID OTPs:

1. Implement the logic to collect the ArcotID OTP from the user.
2. (**Optional**) If you are implementing a plug-in, then invoke the setAdditionalInput() method in the AdditionalInput class to obtain an object that implements the class.

   See "Preparing Additional Input" (see page 92) for more information.
3. Invoke the verifyOTP() method of the ArcotOTPAuth interface to verify the OTP of the user. Optionally, you can also specify the token type that must be returned to the user *after* successful authentication by using the AuthTokenType class.

   This method returns an instance of the AuthResponse interface, which provides the transaction details, credential details, and token information.

ArcotOTP (ArcotOTP-OATH) Synchronization

To synchronize the client and server OATH-compliant ArcotOTP:

1. (**Optional**) If you are implementing a plug-in, then invoke the setAdditionalInput() method in the AdditionalInput class to obtain an object that implements the class.

   See "Preparing Additional Input" (see page 92) for more information.

2. Invoke the synchronizeOTP() method of the ArcotOTPAuth interface to synchronize the server ArcotOTP with the client ArcotOTP. Optionally, you can also specify the token type that must be returned to the user *after* successful authentication by using the AuthTokenType class.

   This method returns an instance of the AuthResponse interface, which provides the transaction details, credential details, and token information.

EMV (ArcotID OTP-EMV) Authentication

To authenticate the ArcotID OTPs that are EMV compliant:

1. Implement the logic to collect the EMV OTP from the user.

2. (**Optional**) If you are implementing a plug-in, then invoke the setAdditionalInput() method in the AdditionalInput class to obtain an object that implements the class.

   See "Preparing Additional Input" (see page 92) for more information.

3. Invoke the verifyOTP() method of the EMVAuth interface to verify the OTP of the user. Optionally, you can also specify the token type that must be returned to the user *after* successful authentication by using the AuthTokenType class.

   This method returns an instance of the AuthResponse interface, which provides the transaction details, credential details, and token information.

EMV (ArcotID OTP-EMV) Password Synchronization

To synchronize the client and server EMV-compliant ArcotID OTP:

1. (**Optional**) If you are implementing a plug-in, then invoke the setAdditionalInput() method in the AdditionalInput class to obtain an object that implements the class.

   See "Preparing Additional Input" (see page 92) for more information.

2. Invoke the synchronizeOTP() method of the EMVAuth interface to synchronize the server OTP with the client OTP. Optionally, you can also specify the token type that must be returned to the user *after* successful authentication by using the AuthTokenType class.

   This method returns an instance of the AuthResponse interface, which provides the transaction details, credential details, and token information.

Verifying Password Type Credentials

The authentication requests that are presented to the AuthMinder Server must specify the type of credential that has to be used to process the requests. In case of RADIUS and ASSP authentication requests, the input requests do not have the provision to specify the type of credential, and by default RADIUS uses One-Time Password and ASSP uses password credential for authentication.

To support any password-based authentication mechanisms for RADIUS and ASSP, or to map any input request with an unknown credential type to a particular password-based authentication mechanism you must create the *Credential Type Resolution* configuration. You can map the input request to any of the following credentials that AuthMinder supports:

- Password
- OTP
- OATH OTP
- ArcotID OTP-OATH
- ArcotID OTP-EMV
- RADIUS OTP
- LDAP Password
- Native Token

If a particular input request uses the credential resolution configuration, then the verifyPlain method in the PlainAuth interface is invoked to process that request. Based on the configuration, the incoming user credential will be mapped to the credential that it is configured to.

Note: To use this feature, you should have configured the created credential type resolution using the Administration Console. Refer to *CA AuthMinder Administration Guide* for more information.

To verify a password type credential:

1. Implement the logic to collect the password from the user.
2. (**Optional**) If you are implementing a plug-in, then invoke the setAdditionalInput() method in the AdditionalInput class to obtain an object that implements the class.

   See "Preparing Additional Input" (see page 92) for more information.
3. Invoke the verifyPlain() method of the PlainAuth interface to verify the password of the user. Optionally, you can also specify the token type that must be returned to the user *after* successful authentication by using the AuthTokenType class.

   This method returns an instance of the AuthResponse interface, which provides the transaction details, credential details, and token information.

Verifying the Authentication Tokens

The AuthMinder Authentication SDK provides an appropriate token to the end user after they authenticate successfully. The token is then presented to the AuthMinder Server, indicating that the user is authenticated and can be provided access to the protected resources.

By using the Authentication Web service, you can specify whether the token has to be returned after authentication or not. In addition, you can also specify the type of the token that must be returned after authentication. The verifyAuthToken method specifies the return token type and supports the following types of tokens:

- **Native Tokens**

  Specify this type when CA-proprietary (or Native) token is required after successful authentication. This token can be used multiple times before it expires.

- **One-Time Tokens**

  Specify this type when one-time token is required after successful authentication. This token can be used only one time before it expires.

- **SAML Tokens**

  Secure Assertion Markup Language (*SAML*) is an open standard, which specifies the format of the authentication data exchanged between security domains. The Native, Default, and One-Time tokens issued by AuthMinder can only be interpreted by the AuthMinder Server, but the SAML tokens issued by the AuthMinder Server can be interpreted by any other authentication system. AuthMinder supports **1.1** and **2.0** versions of SAML:

  - SAML 1.1 Tokens

    Specify this type of token when you are using custom (non-AuthMinder) authentication mechanism that needs SAML 1.1 tokens after successful authentication.

  - SAML 2.0 Tokens

    Specify this type of token when you are using custom (non-AuthMinder) authentication mechanism that needs SAML 2.0 tokens after successful authentication.

- **Default Tokens**

  Specify this type of token when the default token configured at the server is to be requested after successful authentication.

- **Custom**

  Specify this type of token when you are performing custom credential authentication.

AuthMinder Server can verify *only* the Native and One-Time tokens that are issued to the users. The authentication token must be verified in cases when you use the token for Single Sign-On, wherein you authenticate the user once and allow them to use multiple resources using the same authentication token.

To verify if a token is valid or not:

1. (**Optional**) If you are implementing a plug-in, then invoke the setAdditionalInput() method in the AdditionalInput class to obtain an object that implements the class.

   See "Preparing Additional Input" (see page 92) for more information.

2. Invoke the verifyAuthToken() method in Authentication class to verify the token of the user.

   This method returns an instance of the AuthTokenResponse interface, which provides the credential and transaction details.

Fetching the PAM

Personal Assurance Message (*PAM*) is a security feature that reassures the end users that they are accessing the genuine site of your organization, and not a phished site.

Important! For the CA Advanced Authentication out-of-the-box flows, PAM is not enabled. However, this feature is available as a custom option.

To obtain the PAM of a user:

1. (**Optional**) If you are implementing a plug-in, then invoke the setAdditionalInput() method in the AdditionalInput class to obtain an object that implements the class.

   See "Preparing Additional Input" (see page 92) for more information.

2. Invoke the getPAM() method in Authentication class to fetch the PAM of the user.

   This method returns an instance of the PAMResponse interface, which provides the user details, PAM, and transaction details.

Authentication Operations Summary

The following table provides a summary of the input parameters required for performing authentication operations discussed in this chapter:

Operation	Input Required	Expected Output
ArcotID	■ User name (userName) ■ (Optional) Organization name (orgName) **Note:** If the organization name is not provided, then the user is assumed to belong to the default organization. ■ Signed challenge (signedResponse) ■ (Optional) Additional Input (AdditionalInput) ■ (Optional) Authentication token type (AuthTokenType)	■ AuthResponse
QnA	■ User name (userName) ■ (Optional) Organization name (orgName) **Note:** If the organization name is not provided, then the user is assumed to belong to the default organization. ■ Question and Answers for authentication (qnaInfo) ■ (Optional) Additional Input (AdditionalInput) ■ (Optional) Authentication token type (AuthTokenType)	
Password	■ User name (userName) ■ (Optional) Organization name (orgName) **Note:** If the organization name is not provided, then the user is assumed to belong to the default organization. ■ User password for authentication (password) ■ (Optional) Additional Input (AdditionalInput) ■ (Optional) Authentication token type (AuthTokenType)	

Operation	Input Required	Expected Output
OTP	■ User name (userName) ■ (Optional) Organization name (orgName) **Note:** If the organization name is not provided, then the user is assumed to belong to the default organization. ■ One-time password for authentication (otp) ■ (Optional) Additional Input (AdditionalInput) ■ (Optional) Authentication token type (AuthTokenType)	
Password type credential	■ User name (userName) ■ (Optional) Organization name (orgName) **Note:** If the organization name is not provided, then the user is assumed to belong to the default organization. ■ User password for authentication (password) ■ (Optional) Additional Input (AdditionalInput) ■ (Optional) Authentication token type (AuthTokenType)	■ AuthResponse
OATH OTP	■ User name (userName) ■ (Optional) Organization name (orgName) **Note:** If the organization name is not provided, then the user is assumed to belong to the default organization. ■ OATH OTP for authentication (otp) ■ (Optional) Additional Input (AdditionalInput) ■ (Optional) Authentication token type (AuthTokenType)	
ArcotOTP-OATH	■ User name (userName) ■ (Optional) Organization name (orgName) **Note:** If the organization name is not provided, then the user is assumed to belong to the default organization. ■ ArcotOTP for authentication (otp) ■ (Optional) Additional Input (AdditionalInput) ■ (Optional) Authentication token type (AuthTokenType)	

Operation	Input Required	Expected Output
ArcotOTP-EMV	■ User name (userName) ■ (Optional) Organization name (orgName) **Note:** If the organization name is not provided, then the user is assumed to belong to the default organization. ■ ArcotOTP for authentication (otp) ■ (Optional) Additional Input (AdditionalInput) ■ (Optional) Authentication token type (AuthTokenType)	

Appendix A: Input Data Validations

To ensure that the system does not process invalid data, to enforce business rules, and to ensure that user input is compatible with internal structures and schemas, AuthMinder Server validates the data that it receives from the APIs. The following table explains the criteria that the AuthMinder Server uses to validate this input data:

Note: Attribute length mentioned in the following table corresponds to the character length.

Attribute	**Parameter Name**	**Validation Criteria**
Protocol Status	PROTOCL_STATUS	Checks for the following values: ■ PROTOCOL_STATUS_ACTIVE ■ PROTOCOL_STATUS_DISABLED
Port Number	PORT_NUMBER	Length is between 1 and 65535 characters.
Port Type	PORT_TYPE	■ Is non-empty ■ Checks for the following values: ■ TCP ■ SSL ■ UDP
Client Root ID	CLIENT_ROOT_ID	Checks with a set of client root IDs
Server Certificate chain encoding	SERVER_CERT_CHAIN_ENCODING	■ Server certificate chain encoding is non-empty. ■ Checks for the PEM format.
Server Certificate Chain	SERVER_CERT_CHAIN	Server certificate chain is valid.
Client Certificate Chain	CLIENT_CERT_CHAIN	Client certificate chain is valid.
Client Root CA Certificate	CLIENT_ROOT_CA_CERT	Client root CA certificate is valid.
Server Root CA Certificate	SERVER_ROOT_CA_CERT	Server root CA certificate is valid.
Client Root CA Certificates Count	CLIENT_ROOT_CA_CERT	Checks the count of CA certificates is non-zero.
Client Root ID	CLIENT_ROOT_ID	Checks with a set of client root IDs.

Attribute	**Parameter Name**	**Validation Criteria**
Server Certificate Chain Encoding	SERVER_CERT_CHAIN_ENCODING	■ Server certificate chain encoding is non-empty. ■ Checks for the PEM format.
Server Certificate Chain	SERVER_CERT_CHAIN	Server certificate chain is valid.
Client Certificate Chain	CLIENT_CERT_CHAIN	Client certificate chain is valid.
Client Root CA Certificate	CLIENT_ROOT_CA_CERT	Client root CA certificate is valid.
Server Root CA Certificate	SERVER_ROOT_CA_CERT	Server root CA certificate is valid.
Client Root CA Certificate count	CLIENT_ROOT_CA_CERT	Checks the count of CA certificates is non-zero.
Server Private Key Encoding	SERVER_PRIVATE_KEY_ENCODING	■ Server private key encoding is non-empty. ■ Checks for the PEM format.
Locale Name	LOCALE_NAME	■ Locale name is non-empty. ■ Checks locale name with the ISO set of locales.
Client Root CA Path	CLIENT_ROOT_CA_PATH	Client root CA path is non-empty.
Server ID	SERVER_ID	■ Port number > 1. ■ Checks with a set of server identifiers.
Client Root CA Certificate Encoding	CLIENT_ROOT_CA_CERT_ENCODING	■ Client root CA certificate encoding is non-empty. ■ Checks for the PEM format.
Certificate Common Name	CERT_COMMON_NAME	■ Certificate common name is non-empty. ■ Certificate common name length is between 1 and 256. ■ Does not contain invalid characters (ASCII 0-31).

Attribute	Parameter Name	Validation Criteria
Certificate Country Name	COUNTRY_NAME	■ Certificate country name is non-empty. ■ Certificate country name length is between 1 and 64. ■ Does not contain invalid characters (ASCII 0-31).
Certificate Organization Name	ORG_NAME	■ Certificate organization name is non-empty. ■ Certificate organization name length is between 1 and 64. ■ Does not contain invalid characters (ASCII 0-31).
Certificate Organization Unit Name	ORG_UNIT_NAME	■ Certificate organization unit name is non-empty. ■ Certificate organization unit name length is between 1 and 64. ■ Does not contain invalid characters (ASCII 0-31).
Certificate State Name	STATE_NAME	■ Certificate state name is non-empty. ■ Certificate state name length is between 1 and 64. ■ Does not contain invalid characters (ASCII 0-31).
Certificate Locality Name	LOCALITY_NAME	■ Certificate locality name is non-empty. ■ Certificate locality name length is between 1 and 64. ■ Does not contain invalid characters (ASCII 0-31).
Certificate Start Date	START_TIME	Checks for valid date format.
Certificate End Date	END_TIME	Checks for valid date format.
PKI Certificate	PKI_CERTIFICATE	PKI certificate is valid.
PKI Key	PKI_KEY	PKI key is valid.

Attribute	**Parameter Name**	**Validation Criteria**
Certificate Chain and Key Pair	PRIVATE_KEY_PAIR	Certificate chain and key pair are valid.
PKCS12 Certificate Chain	PKCS12_CERT_CHAIN_KEY	PKCS12 certificate chain is valid.
PKCS7 Certificate Chain	PKCS12_CERT_CHAIN_KEY	PKCS7 certificate chain is valid.
User ID	USER_ID	Minimum value of user ID must be greater than 1.
Group ID	GROUP_ID	Minimum value of group ID must be greater than 1.
Create Time	CREATE_TIME	Checks for valid date format.
Last Modified Time	LAST_MODIFIED_TIME	Checks for valid date format.
Start and End Date	START_END_DATES	Start date < End date.
User Attribute Name	USER_ATTR_NAME	User Attribute Name is non-empty.
WebFort organization name (checks for the organization name is '\n', else validate)	ORG_NAME	■ Organization name is non-empty. ■ Organization name length is between 1 and 64. ■ Does not contain invalid characters (ASCII 0-31).
User Existence Check	USER_EXISTENCE_CHECK	Value of user existence check is 0 or 1.
User Active Check	USER_ACTIVE_CHECK	Value of user active check is 0 or 1.
Kerberos User Name	KERBEROS_USER_NAME	■ Kerberos user name is non-empty. ■ Kerberos user name length is between 1 and 64. ■ Does not contain invalid characters (ASCII 0-31).
Kerberos Domain Name	KERBEROS_DOMAIN_NAME	■ Kerberos domain name is non-empty. ■ Kerberos domain name length is between 1 and 64. ■ Does not contain invalid characters (ASCII 0-31).

Attribute	Parameter Name	Validation Criteria
Kerberos Password	KERBEROS_PASSWORD	■ Kerberos password is non-empty. ■ Kerberos password length is between 1 and 64. ■ Does not contain invalid characters (ASCII 0-31).
Authentication User Password	AUTH_USER_PASSWORD	■ User password is non-empty. ■ User password is between 1 and 64. ■ Checks user password against to a set of strings. ■ Does not contain invalid characters (ASCII 0-31).
Password Maximum Length	PWD_MAX_LENGTH	■ Minimum value of password maximum length must be greater than 4. ■ Maximum value of password maximum length must be less than 64.
Password Minimum Length	PWD_MIN_LENGTH	■ Minimum value of password minimum length must be greater than 4. ■ Maximum value of password minimum length must be less than 64.
Password Minimum Special Character Length	PWD_SPECIAL_CHAR_MIN_LENGTH	■ Minimum value of password special character length must be greater than 0. ■ Maximum value of password special character minimum length must be less than 64.
Password Minimum Alphabetic Character Length	PWD_ALPHA_CHAR_MIN_LENGTH	■ Minimum value of password alphabetic character length must be greater than 0. ■ Maximum value of password alphabetic character length must be less than 64.

Attribute	Parameter Name	Validation Criteria
Password Minimum Numeric Character Length	PWD_NUMERIC_CHAR_MIN_LENGTH	■ Minimum value of password numeric character length must be greater than 0. ■ Maximum value of password numeric character length must be less than 64.
Password Strength Configuration	PASSWORD_STRENGTH	Password strength attribute length must be less than the password length.
Question	AUTH_QUESTIONS	■ Question is non-empty. ■ Question length is between 1 and 64. ■ Does not contain invalid characters (ASCII 0-31).
Answer	AUTH_ANSWERS	■ Answer is non-empty. ■ Answer length is between 1 and 64. ■ Does not contain invalid characters (ASCII 0-31).
Number of Questions	NUM_OF_QNA	■ Number of questions must be greater than the minimum number of questions. ■ Number of questions must be lesser than the maximum number of questions.
Number of Questions to Ask	QNA_NUM_QUESTION_TO_ASK	■ Minimum questions to ask must be greater than 1. ■ Maximum questions to ask must be lesser than 10.
Minimum Number of Correct Answers Required	QNA_MIN_ANS_REQUIRED	■ Minimum correct answers must be greater than 1. ■ Minimum correct answers must be less than 10.
QnA Maximum Questions	MAX_QUESTIONS	■ Minimum value of maximum questions must be greater than 1. ■ Maximum value of maximum questions must be less than 10.

Attribute	**Parameter Name**	**Validation Criteria**
QnA Minimum Questions	MIN_QUESTIONS	■ Minimum value of minimum questions must be greater than 2. ■ Maximum value of minimum questions must be less than 10.
QnA Challenge Timeout in Seconds	QNA_CHALLENGE_TIMEOUT_SECS	QnA challenge timeout in seconds must be between 1 and 7200.
Plain Key Type	PLAIN_KEY_TYPE	■ Plain key type is non-empty. ■ Checks for the RSA value.
Arcot Key Type	ARCOT_KEY_TYPE	■ Plain key type is non-empty. ■ Checks for the RSA value.
Plain Key Length	PLAIN_KEY_LENGTH	Plain key length value must be between 512 and 4096.
Arcot Key Length	ARCOT_KEY_LENGTH	Arcot key length is between 512 and 4096.
ArcotID Challenge Timeout in Seconds	ARCOTID_CHALLENGE_TIMEOUT_SECS	The ArcotID PKI challenge timeout in seconds is between 1 and 7200.
ArcotID Unsigned Attribute Key Check	AID_UNSIGNED_ATTRIB_KEY	Unsigned attribute key is either USERID or ORG.
Warning Period in Days	WARNING_PERIOD_DAYS	Warning period in days is greater than 0.
Grace Period in Days	GRACE_PERIOD_DAYS	Grace period in days is greater than 0.
Auto Unlock Period in Hours	AUTO_UNLOCK_PERIOD_HOURS	Auto-unlock period in hours is greater than 0.
Authentication OTT Token	AUTH_OTT_TOKEN	■ OTT token is non-empty. ■ OTT token length is between 4 and 64.
OTT Length	OTT_LENGTH	Value of OTT length is between 5 and 240.
OTT Timeout in Seconds	OTT_TIMEOUT	Value of OTT timeout in seconds is between 1 and 172800.
OTP Length	OTP_LENGTH	Value of OTP length is between 4 and 64.

Attribute	Parameter Name	Validation Criteria
OTP Type	OTP_TYPE	Checks for numeric and alphanumeric values.
OTP Multiple Usage Count	OTP_MULTIPLE_USAGE_COUNT	Multiple usage count of OTP is between 1 and 99999.
Global Authentication Token Timeout in Seconds	GLOBAL_AUTH_TOKEN_TIMEOUT_SECS	Global authentication token timeout in seconds is between 1 and 172800.
Maximum Strikes	MAX_STRIKES	Maximum strike count is between 1 and 100.
Transaction Algorithm ID	TRANSALGO_ID	Checks for the following values: ■ NATIVE_PLAIN_CS ■ NATIVE_PLAIN_CI ■ NATIVE_SHA1_CS ■ NATIVE_SHA1_CI
Organization Credential Configuration Name	ORG_CRED_CONFIG_NAME	Organization credential configuration name is non-empty.
ArcotID Credential Configuration Name	ARCOTID_CRED_CONFIG_NAME	■ ArcotID PKI credential configuration name is non-empty. ■ Checks ArcotID PKI credential configuration name with a set of strings. ■ ArcotID PKI credential configuration name length is between 1 and 64. ■ Does not contain invalid characters (ASCII 0-31).
OTP Credential Configuration Name	OTP_CRED_CONFIG_NAME	■ OTP credential configuration name is non-empty. ■ Checks OTP credential configuration name against to a set of strings. ■ OTP credential configuration name length is between 1 and 64 ■ Does not contain invalid characters (ASCII 0-31).

Attribute	**Parameter Name**	**Validation Criteria**
QnA Credential Configuration Name	QNA_CRED_CONFIG_NAME	■ QnA credential configuration name is non-empty. ■ Checks QnA credential configuration name with a set of strings. ■ QnA credential configuration name length is between 1 and 64. ■ Does not contain invalid characters (ASCII 0-31).
Password Credential Configuration Name	UP_CRED_CONFIG_NAME	■ Password credential configuration name is non-empty. ■ Checks Password credential configuration name with a set of strings. ■ Password credential configuration name length is between 1 and 64 ■ Does not contain invalid characters (ASCII 0-31).
ArcotID Authentication Policy Name	ARCOTID_AUTH_POLICY_NAME	■ ArcotID PKI authentication policy name is non-empty. ■ Checks ArcotID PKI authentication policy name with a set of strings. ■ ArcotID PKI authentication policy name length is between 1 and 64 ■ Does not contain invalid characters (ASCII 0-31).
OTP Authentication Policy Name	OTP_AUTH_POLICY_NAME	■ OTP authentication policy name is non-empty. ■ Checks OTP authentication policy name with a set of strings. ■ OTP authentication policy name length is between 1 and 64. ■ Does not contain invalid characters (ASCII 0-31).

Attribute	Parameter Name	Validation Criteria
QnA Authentication Policy Name	QNA_AUTH_POLICY_NAME	■ QnA authentication policy name is non-empty. ■ Checks QnA authentication policy name with a set of strings. ■ QnA authentication policy name length is between 1 and 64. ■ Does not contain invalid characters (ASCII 0-31).
Password Authentication Policy Name	PASSWORD_AUTH_POLICY_NAME	■ Password authentication policy name is non-empty. ■ Checks Password authentication policy name with a set of strings. ■ Password authentication policy name length is between 1 and 64. ■ Does not contain invalid characters (ASCII 0-31).
General Authentication Policy Name	GENERAL_AUTH_POLICY_NAME	■ General authentication policy name is non-empty. ■ Checks General authentication policy name with a set of strings. ■ General authentication policy name length is between 1 and 64. ■ Does not contain invalid characters (ASCII 0-31).
RADIUS Authentication Policy Name	RADIUS_AUTH_POLICY_NAME	■ RADIUS authentication policy name is non-empty. ■ Checks RADIUS authentication policy name with a set of strings. ■ RADIUS authentication policy name length is between 1 and 64 ■ Does not contain invalid characters (ASCII 0-31).

Attribute	Parameter Name	Validation Criteria
Kerberos Authentication Policy Name	KERBEROS_AUTH_POLICY_NAME	■ Kerberos authentication policy name is non-empty. ■ Checks Kerberos authentication policy name with a set of strings. ■ Kerberos authentication policy name length is between 1 and 64. ■ Does not contain invalid characters (ASCII 0-31).
Mechanism Name	MECHANISM_NAME	■ Mechanism name is non-empty. ■ Does not contain invalid characters (ASCII 0-31). ■ Checks mechanism name with a set of strings.
Mechanism Status	MECHANISM_STATUS	Checks for the following values: ■ MECHANISM_STATUS_ENABLE ■ MECHANISM_STATUS_DISABLED
Radius Client IP Address	RADIUS_CLIENT_IP	■ Radius client IP address is non-empty. ■ Radius client IP address length is between 7 and15. ■ Does the following checks: ■ It should contain integers and '.' ■ It should contain three dots
Radius Client Shared Secret	RADIUS_ClIENT_SHARED_SECRET	■ Radius client shared secret is non-empty. ■ Radius client shared secret length is between 1 and 1024.
Radius Client Description	RADIUS_CLIENT_DESC	■ Radius client description length is between 0 and 256. ■ Does not contain invalid characters (ASCII 0-31).

Attribute	Parameter Name	Validation Criteria
Radius Client Authentication Type	RADIUS_CLIENT_AUTH_TYPE	■ Radius client shared secret is non-empty. ■ Checks for the following values: ■ OTT ■ INBAND
Radius Client Maximum Chunk Size	RADIUS_CLIENT_MAX_CHUNK_SIZE	RADIUS client maximum chunk size is between 50 and 200.
Radius Version	RADIUS_VERSION	Checks for the following values: ■ 1 ■ 2
Duplicate Question and Answers	DUPLICATE_QUESTION_AND_ANSWER	■ Questions are not duplicate. ■ Answers are not duplicate. ■ Question is not same as answer.
Token Type	AUTH_TOKEN_TYPE	Checks for the following values: ■ DEFAULT_TOKEN ■ NATIVE_TOKEN ■ OTP_TOKEN ■ SAML11_TOKEN ■ SAML20_TOKEN ■ NO_TOKEN
Configuration Name	CONFIG_NAME	■ Configuration name is non-empty. ■ Configuration name length is between 1 and64. ■ Does not contain invalid characters (ASCII 0-31).
Pin	PIN	■ Pin is non-empty. ■ Pin length is between 1 and 64. ■ Does not contain invalid characters (ASCII 0-31).
OTP Maximum Length	OTP_MAX_LENGTH	OTP maximum length is between 4 and 64.
OTP Minimum Length	OTP_MIN_LENGTH	OTP minimum length is between 4 and 64.

Attribute	**Parameter Name**	**Validation Criteria**
Last Strike Time	LAST_STRIKE_TIME	Checks for valid date format.
Last Failed Time	LAST_FAILED_TIME	Checks for valid date format.
Last Succeeded Time	LAST_SUCCEEDED_TIME	Checks for valid date format.
Credential Status	CRED_STATUS	Checks for the following values: ■ ACTIVE ■ LOCKED ■ DISABLED ■ REVOKED ■ REISSUED ■ VERIFIED
Certificate Serial Number	CERT_SERIAL_NUMBER	■ Certificate serial number is non-empty. ■ Certificate serial number length is between 1 and32. ■ Checks for the following characters: ■ 0 – 9 ■ a – f ■ A - F
Password Minimum and Maximum Length	PWD_MIN_LENGTH	Password minimum length is lesser than password maximum length.
QnA Minimum and Maximum Questions	MIN_QUESTIONS	QnA minimum questions is lesser than QnA maximum questions.
Questions and Correct Answers	QNA_NUM_QUESTION_TO_ASK	Number of correct answers is lesser than number of questions.
Host Name	HOST_NAME	■ Host name is non-empty. ■ Host name length is between 1 and 64 ■ Does not contain invalid characters (ASCII 0-31).

Attribute	**Parameter Name**	**Validation Criteria**
URI	URI_NAME	■ URI is non-empty. ■ URI length is between 1 and 1024. ■ Does not contain invalid characters (ASCII 0-31).
Connection Timeout	CONNECTION_TIMEOUT	Connection timeout is between 0 and 2147483647.
Read Timeout	READ_TIMEOUT	Read timeout is between 0 and 2147483647.
Idle Timeout	IDLE_TIMEOUT	Idle timeout is between 0 and 2147483647.
Minimum Connections	MIN_CONNECTIONS	Minimum connections is between 0 and 2147483647.
Maximum Connections	MAX_CONNECTIONS	Maximum connections is between 0 and 2147483647.
WebFort Event ID	WF_EVENT_ID	Checks for the set of valid events.
Instance name	INSTANCE_NAME	■ Instance name is non-empty. ■ Instance name length is between 1 and 64. ■ Does not contain invalid characters (ASCII 0-31).
Log Level	LOG_TXN_LOG_LEVEL	Minimum database connections is between 1 and 3.
Minimum DB Connections	MIN_DB_CONNECTIONS	Minimum database connections is between 1 and 128.
Maximum DB Connections	MAX_DB_CONNECTIONS	Maximum database connections is between 1 and 512.
Maximum DB Connections Against Minimum	MAX_DB_CONNECTIONS	Maximum database connections are less than minimum database connections.
Increment DB Connections	INC_DB_CONNECTIONS	■ Increment database connections must be greater than 0. ■ Increment database connections must be less than maximum database connections-minimum database connections.

Attribute	**Parameter Name**	**Validation Criteria**
ArcotID Unsigned Attribute Key (No validation on value)	AID_UNSIGNED_ATTRIB_KEY	Attributes with name USERID and ORG are not allowed because these are created while creating ArcotID PKI. Therefore, these values cannot be modified.
Custom Attributes	NOTES_KEY/ NOTES_VALUE/ NOTES	■ Does not contain invalid characters (ASCII 0-31). ■ Custom attribute string length must be between 0 and 1024.
SSL Trust Store Group Name	SSL_TRUST_STORE_GROUP_NAME	■ SSL trust store group name is non-empty. ■ SSL trust store group name length is between 1 and 64. ■ Does not contain invalid characters (ASCII 0-31).
Minimum Threads	MIN_THREADS	Minimum thread count is between 1 and 1024.
Maximum Threads	MAX_THREADS	Maximum thread count is between 1 and 1024.
Threads Minimum and Maximum Count	MIN_THREADS	Minimum thread count is less than maximum thread count.
Additional Input	ADDITIONAL_INPUTS_NAME	Does not contain invalid characters (ASCII 0-31).
Server Statistics Option	STATS_OPTION	Checks for the following values: ■ CONSOLIDATED ■ PER_PROTOCOL ■ DATABASE ■ UDS_CLIENT ■ MAXVAL
Numeric Instance Attribute	parameterName that is passed to the function	Checks only if the numeric instance attributes are used.

Attribute	**Parameter Name**	**Validation Criteria**
Display Name	DISPLAY_NAME	■ Display name is non-empty. ■ Display name length is between 0 and 256. ■ Does not contain invalid characters (ASCII 0-31).
Logo URL	LOGO_URL	Checks if the URL format is valid.
Password Challenge Validity	PASSWORD_CHALLENGE_TIMEOUT_SECS	Password challenge validity is between 1 and 7200.
ArcotID Card Name	AUTH_CARD_NAME	■ ArcotID PKI Card Name is non-empty. ■ ArcotID PKI Card Name length is between 1 and 8.
Duplicate Questions	DUPLICATE_QUESTIONS	Questions are not duplicate.
Duplicate Answers	DUPLICATE_ANSWERS	Answers are not duplicate.
Partial password Length	PARTIAL_PWD_LENGTH	Partial password length is between 0 and 64.
QnA Shuffle Mode	QNA_SHUFFLE_MODE	Checks for the following values: ■ RANDOM ■ ALTERNATIVE
QnA Shuffle Flag	QNA_SHUFFLE_FLAG	Checks for the following values: ■ SHUFFLE_ALWAYS ■ SHFFULE_AFTER_SUCCESS_AUTH
QnA Return Mode	QNA_RETURN_MODE	Checks for the following values: ■ STATIC ■ RANDOM
OATH One-Time Password Length	OATH_OTP_LENGTH	OATH One-Time Password length is between 4 and 32.
OATH One-Time Password Token Type	OATH_OTP_TYPE	Checks for the following values: ■ HOTP ■ TOTP
OATH One-Time Password Authentication Look Ahead Count	OATH_OTP_AUTH_LOOK_AHEAD	OATH One-Time Password Authentication look ahead count is between 0 and 99999.

Attribute	Parameter Name	Validation Criteria
OATH One-Time Password Authentication Look Back Count	OATH_OTP_AUTH_LOOK_BACK	OATH One-Time Password Authentication look back count is between 0 and 99999.
OATH One-Time Password Synchronization Look Ahead Count	OATH_OTP_RESYNC_LOOK_AHEAD	OATH One-Time Password Synchronization look ahead count is between 0 and 99999.
OATH One-Time Password Synchronization Look Back Count	OATH_OTP_RESYNC_LOOK_BACK	OATH One-Time Password Synchronization look back count is between 0 and 99999.

Appendix B: AuthMinder Logging

To effectively manage the communication between AuthMinder Server and your application, it is necessary to get information about the activity and performance of the Server as well as any problems that have occurred.

This appendix describes the various log files supported by AuthMinder, the severity levels that you will see in these files, and the formats of these log files. It covers the following topics:

About the Log Files

The AuthMinder log files can be categorized as:

- Installation Log File (see page 127)
- AuthMinder Server Startup Log File (see page 127)
- AuthMinder Server Log File (see page 128)
- UDS Log File (see page 129)
- Administration Console Log File (see page 130)

The parameters that control logging in these files can be configured either by using the relevant INI files (as is the case with Administration Console, UDS, and AuthMinder Server startup log files) or by using the Administration Console itself (as is the case with AuthMinder log file.) The typical logging configuration options that you can change in these files include:

- **Specifying log file name and path:** AuthMinder enables you to specify the directory for writing the log files and storing the backup log files. Specifying the diagnostic logging directory allows administrators to manage system and network resources.
- **Log file size:** The maximum number of bytes the log file can contain. When the log files reach this size, a new file is created and the old file is moved to the backup directory.
- **Using log file archiving:** As AuthMinder components run and generate diagnostic messages, the size of the log files increases. If you allow the log files to keep increasing in size, then the administrator must monitor and clean up the log files manually. AuthMinder enables you to specify configuration options that limit how much log file data is collected and saved. AuthMinder lets you specify the configuration option to control the size of diagnostic logging files. This lets you determine a maximum size for the log files. When the maximum size is reached, older log information is moved to the backup file before the newer log information is saved.
- **Setting logging levels:** AuthMinder also allows you to configure logging levels. By configuring logging levels, the number of messages saved to diagnostic log files can be reduced. For example, you can set the logging level so that the system only reports and saves critical messages. See "Supported Severity Levels" (see page 132) for more information on the supported log levels.
- **Specifying time zone information:** AuthMinder enables you to either use the local time zone for time stamping the logged information or use GMT for the same.

Installation Log File

When you install AuthMinder, the installer records all the information that you supply during the installation and the actions (such as creating the directory structure and making registry entries) that it performs in the Arcot_WebFort_Install_*[assign the value for mm in your book]_<dd>_<yyyy>_<hh>_[assign the value for mm in your book]_SpectroSERVER*.log file. The information in this file is very useful in identifying the source of the problems if the AuthMinder installation did not complete successfully.

The default location of this file is:

Windows:

<*install_location*>\

UNIX-Based Platforms:

<*install_location*>/

AuthMinder Server Startup Log File

When you start the AuthMinder Server, it records all start-up (or boot) actions in the arcotwebfortstartup.log file. The information in this file is very useful in identifying the source of the problems if the AuthMinder service does not start up.

The default location of this file is:

Windows:

<*install_location*>\Arcot Systems\logs\

UNIX-Based:

<*install_location*>/arcot/logs/

AuthMinder Server Log File

When you perform AuthMinder Server configurations for example, protocol configurations, profile configurations, policy configurations, and authenticate users, such configurations are written to the arcotwebfort.log file. The default location of this file is:

Windows:

<*install_location*>\Arcot Systems\logs\

UNIX-Based:

<*install_location*>/arcot/logs/

The parameters that control logging in this file can be configured by using the Administration Console. To do so, you must use the instance-specific configuration sub-screen that you can access by clicking the required instance in the **Instance Management** screen.

In addition to the log file path, the maximum log file size (in bytes), backup directory, logging level, and timestamp information, you can also control whether you want to enable trace logging. See section, "Format of the AuthMinder Log Files" (see page 131) for the details of the default format used in the file.

UDS Log File

All User Data Service (UDS) information and actions are recorded in the arcotuds.log file. This information includes:

- UDS database connectivity information
- UDS database configuration information
- UDS instance information and the actions performed by this instance

The information in this file is very useful in identifying the source of the problems if the Administration Console could not connect to the UDS instance. The default location of this file is:

Windows:

<*install_location*>\Arcot Systems\logs\

UNIX-Based:

<*install_location*>/arcot/logs/

The parameters that control logging in this file can be configured by using the udsserver.ini file, which is available in the conf folder in ARCOT_HOME.

In addition to the logging level, log file name and path, the maximum file size (in bytes), and archiving information, you can also control the layout of the logging pattern for UDS by specifying the appropriate values for log4j.appender.debuglog.layout.ConversionPattern. See section, "Format of UDS and Administration Console Log Files" (see page 132) for details of the default format used in the file.

Administration Console Log File

When you deploy the Administration Console and subsequently start it, the details of all its actions and processed requests are recorded in the arcotadmin.log file. This information includes:

- Database connectivity information
- Database configuration information
- Instance information and the actions performed by this instance
- UDS configuration information
- Other Administration Console information specified by the Master Administrator, such as cache refresh

The information in this file is very useful in identifying the source of the problems if the Administration Console does not start up. The default location of this file is:

Windows:

<*install_location*>\Arcot Systems\logs\

UNIX-Based:

<*install_location*>/arcot/logs/

The parameters that control logging in this file can be configured by using the adminserver.ini file, which is available in the conf folder in ARCOT_HOME.

In addition to the logging level, log file name and path, the maximum log file size (in bytes), and log file archiving information, you can also control the layout of the logging pattern for the console by specifying the appropriate values for log4j.appender.debuglog.layout.ConversionPattern. See section, "Format of UDS and Administration Console Log Files" (see page 132) for the details of the default format used in the file.

Format of the AuthMinder Log Files

The following table describes the format of the entries in the following AuthMinder loggers:

- arcotwebfort.log (AuthMinder Server Log File (see page 128))
- arcotwebfortstartup.log (AuthMinder Server Startup Log File (see page 127))

Column	Description
Time Stamp	The time when the entry was logged, translated to the time zone you configured. The format of logging this information is: mm/dd/yy HH:MM:SS.mis Here, mis represents milliseconds.
Log Level (LEVEL) (or Severity)	The severity level of the logged entry. See"Supported Severity Levels" (see page 132) for more information. **Note:** AuthMinder also provides trace logging, which contains the flow details. The trace logs are logged in the arcotwebfort.log file. The entries for the trace messages start with TRACE:.
Protocol Name (PROTOCOLNAME)	The protocol used for the transaction. Possible values are: ■ AUTH_NATIVE ■ ADMIN_WS ■ ASSP_WS ■ RADIUS ■ SVRMGMT_WS ■ TXN_WS In case the server is starting up, shutting down, or is in the monitoring mode, then no protocol is used and the following values are displayed, respectively: ■ STARTUP ■ SHUTDOWN ■ MONITOR
Thread ID (THREADID)	The ID of the thread that logged the entry.
Transaction ID (000TXNID)	The ID of the transaction that logged the entry.
Message	The message logged by the Server in the log file in the free-flowing format. **Note:** The granularity of the message depends on the **Log Level** that you set in the log file.

Format of UDS and Administration Console Log Files

The following table describes the format of the entries in the following loggers:

- arcotuds.log (UDS Log File (see page 129))
- arcotadmin.log (Administration Console Log File (see page 130))

Column	**Associated Pattern** **(In the Log File)**	**Description**
Time Stamp	%d{yyyy-MM-dd hh:mm:ss,SSS z} :	The time when the entry was logged. This entry uses the application server time zone. The format of logging this information is: yyyy-MM-dd hh:mm:ss,SSS z Here, SSS represents milliseconds.
Thread ID	[%t] :	The ID of the thread that logged the entry.
Log Level (or Severity)	%-5p :	The severity level of the logged entry. See Supported Severity Levels (see page 132) for more information.
Logger Class	%-5c{3}(%L) :	The name of the logger that made the log request.
Message	%m%n :	The message logged by the Server in the log file in the free-flowing format. **Note:** The granularity of the message depends on the **Log Level** that you set in the log file.

Refer to the following URL for customizing the **PatternLayout** parameter in the UDS and Administration Console log files:

http://logging.apache.org/log4j/1.2/apidocs/org/apache/log4j/PatternLayout.html

Supported Severity Levels

A *log level* (or *severity level*) enables you to specify the level of detail of the information stored in the AuthMinder logs. This also enables you to control the rate at which the log file will grow.

Server Log File Security Levels

The following table describes the log levels that you see in all log files, in the *decreasing* order of severity:

Log Level		**Description**
0	FATAL	Use this log level for serious, non-recoverable errors that can cause the abrupt termination of the AuthMinder service.
1	WARNI NG	Use this log level for undesirable run-time exceptions, potentially harmful situations, and recoverable problems that are not yet FATAL.
2	INFO	Use this log level for capturing information on run-time events. In other words, this information highlights the progress of the application, which might include changes in: ■ Server state, such as start, stop, and restart. ■ Server properties. ■ State of services. ■ State of a processes on the Server.
3	DEBUG	Use this log level for logging detailed information for debugging purposes. This might include process tracing and changes in Server states.

Note: For AuthMinder Server (arcotwebfort.log) you can set the logging to any of these levels and also enable TRACE logging to capture the flow details.

Note: When you specify a log level, messages from all other levels of *higher* significance are reported as well. For example if the LogLevel is specified as 3, then messages with log levels of FATAL, WARNING, and INFO level are also captured.

Administration Console and UDS Log File Severity Levels

The following table describes the log levels that you see in Administration Console and UDS log files, in the *decreasing* order of severity:

Log Level		**Description**
0	OFF	Use this level to disable all logging.
1	FATAL	Use this log level for serious, non-recoverable errors that can cause the abrupt termination of Administration Console or UDS.
2	WARNIN G	Use this log level for undesirable run-time exceptions, potentially harmful situations, and recoverable problems that are not yet FATAL.

Log Level		**Description**
3	ERROR	Use this log level for recording error events that might still allow the application to continue running.
4	INFO	Use this log level for capturing information on run-time events. In other words, this information highlights the progress of the application, which might include changes in: ■ Server state, such as start, stop, and restart. ■ Server properties. ■ State of services. ■ State of a processes on the Server.
5	TRACE	Use this log level for capturing finer-grained informational events than DEBUG.
6	DEBUG	Use this log level for logging detailed information for debugging purposes. This might include process tracing and changes in Server states.
7	ALL	Use this log level to enable all logging.

Note: When you specify a log level, messages from all other levels of *higher* significance are reported as well. For example if the LogLevel is specified as 4, then messages with log levels of FATAL, WARNING, ERROR, and INFO are also captured.

Sample Entries for Each Log Level

The following subsections show a few sample entries (based on the Log Level) in the **WebFort log file.**

FATAL

```
07/17/09 11:49:20.404  FATAL  STARTUP  00002872  00WFMAIN - Unable to initialize the database

07/17/09 11:49:20.405  FATAL  STARTUP  00002872  00WFMAIN - Failed to load the ini parameters

07/17/09 11:49:20.406  FATAL  STARTUP  00002872  00WFMAIN - Cannot continue due to setConfigData
failure, SHUTTING DOWN
```

WARNING

```
07/17/09 12:50:05.848  INFO  AUTH_NATIVE  00002780  00022508 - Fail to connect to Database prdsn for 1
time(s). DbUsername system

07/17/09 12:50:05.848  INFO  AUTH_NATIVE  00002780  00022508 - ReportError: SQL Error State:08001,
Native Error Code: FFFFFFFF, ODBC Error: [Arcot Systems][ODBC Oracle Wire Protocol
driver][Oracle]TNS-12505: TNS:listener could not resolve SID given in connect descriptor
```

INFO

```
07/17/09 11:51:20.166  INFO  MONITOR  00000424  STATSMON - mMinConnections [4]
07/17/09 11:51:20.166  INFO  MONITOR  00000424  STATSMON - mMaxConnections [128]
07/17/09 11:51:20.166  INFO  MONITOR  00000424  STATSMON - mCurrPoolSize   [4]
07/17/09 11:51:20.166  INFO  MONITOR  00000424  STATSMON - mNumDBFailure     [0]
07/17/09 11:51:20.166  INFO  MONITOR  00000424  STATSMON - mCurrNumUsed      [0]
07/17/09 11:51:20.166  INFO  MONITOR  00000424  STATSMON - mCurrNumAvailable [4]
07/17/09 11:51:20.166  INFO  MONITOR  00000424  STATSMON - idx [0] mNumTimesConnIdxLocked   [24]
07/17/09 11:51:20.166  INFO  MONITOR  00000424  STATSMON - idx [0] mNumTimesConnIdxReleased [24]
07/17/09 11:51:20.166  INFO  MONITOR  00000424  STATSMON - idx [1] mNumTimesConnIdxLocked   [24]
07/17/09 11:51:20.166  INFO  MONITOR  00000424  STATSMON - idx [1] mNumTimesConnIdxReleased [24]
07/17/09 11:51:20.166  INFO  MONITOR  00000424  STATSMON - idx [2] mNumTimesConnIdxLocked   [24]
07/17/09 11:51:20.166  INFO  MONITOR  00000424  STATSMON - idx [2] mNumTimesConnIdxReleased [24]
07/17/09 11:51:20.166  INFO  MONITOR  00000424  STATSMON - idx [3] mNumTimesConnIdxLocked   [23]
07/17/09 11:51:20.166  INFO  MONITOR  00000424  STATSMON - idx [3] mNumTimesConnIdxReleased [23]
07/17/09 11:51:20.166  INFO  MONITOR  00000424  STATSMON - ---------- logging stats for databse [wf-test-p] :
[primary] [ACTIVE] end   ----------
```

DEBUG

```
03/25/10 15:29:30.921 DEBUG SVRMGMT_WS   00000536 00000620 -
ArDBPoolManagerImpl::getLockedDBConnection: [primary] DSN [webfort] is active. Will get the connection from
this
```

```
03/25/10 15:29:30.921 DEBUG SVRMGMT_WS   00000536 00000620 -
ArDBPoolManagerImpl::getLockedDBConnection: Returning DBPool [0112FD80]
03/25/10 15:29:30.921 DEBUG SVRMGMT_WS   00000536 00000620 - ArDBM::Number of queries being
executed [1]
03/25/10 15:29:30.921 DEBUG SVRMGMT_WS   00000536 00000620 - ArDBM::Found query string for query-id :
[SSL_TRUST_STORE_FETCH_ALL].
03/25/10 15:29:30.921 DEBUG SVRMGMT_WS   00000536 00000620 - ArDBM::Executing
Query[ArWFSSLTrustStoreQuery_FetchAll]
03/25/10 15:29:30.921 DEBUG SVRMGMT_WS   00000536 00000620 - Number of rows fetched : 0
```

(For WebFort Server *Only*) Trace Logs

```
03/25/10 15:23:38.515 DEBUG SVRMGMT_WS   00004396 00000596 - TRACE: Released Cache read lock on
[01129D98]
03/25/10 15:23:38.515 DEBUG SVRMGMT_WS   00004396 00000596 - TRACE: CallTrace::Leaving :
[ArDBPoolManagerImpl::selectAnActivePool]. time : 0
03/25/10 15:23:38.515 DEBUG SVRMGMT_WS   00004396 00000596 - TRACE: CallTrace::Entering :
[ArDBPool::getLockedDBConnectionConst]
03/25/10 15:23:38.515 DEBUG SVRMGMT_WS   00004396 00000596 - TRACE:
ArDBPool::getLockedDBConnection [(primary)] : GotContext [1], [3] more connections available
03/25/10 15:23:38.515 DEBUG SVRMGMT_WS   00004396 00000596 - TRACE: CallTrace::Leaving :
[ArDBPool::getLockedDBConnectionConst]. time : 0
```

Appendix C: Additional Settings

This appendix discusses the following topics:

- Configuring Multiple AuthMinder Server Instances (see page 137)
- Setting up SSL (see page 138)

Configuring Multiple AuthMinder Server Instances

To configure Java SDKs with multiple AuthMinder Server instances, you must edit the properties file. By default, the file provides entries to configure 1 AuthMinder Server instance. These entries are appended with 1, indicating that only one server is configured. Depending on the number of instances you want to configure, you must duplicate these entries and append the instance number accordingly.

Perform the following steps to configure AuthMinder Server instance:

1. Depending on the SDK you are configuring, open the respective properties file available in the following folder:

   Windows:
   <*install_location*>\Arcot Systems\sdk\client\java\properties

   Unix-Based Platforms:
   <*install_location*>/arcot/sdk/client/java/properties

2. Set the value of the transport.<*n*> parameter to the required communication mode. By default, it is set to TCP, see "Setting up SSL" (see page 138) if you want to change the communication mode.

3. Set the value of host.<*n*> parameter to the host name or the IP address of the AuthMinder Server.

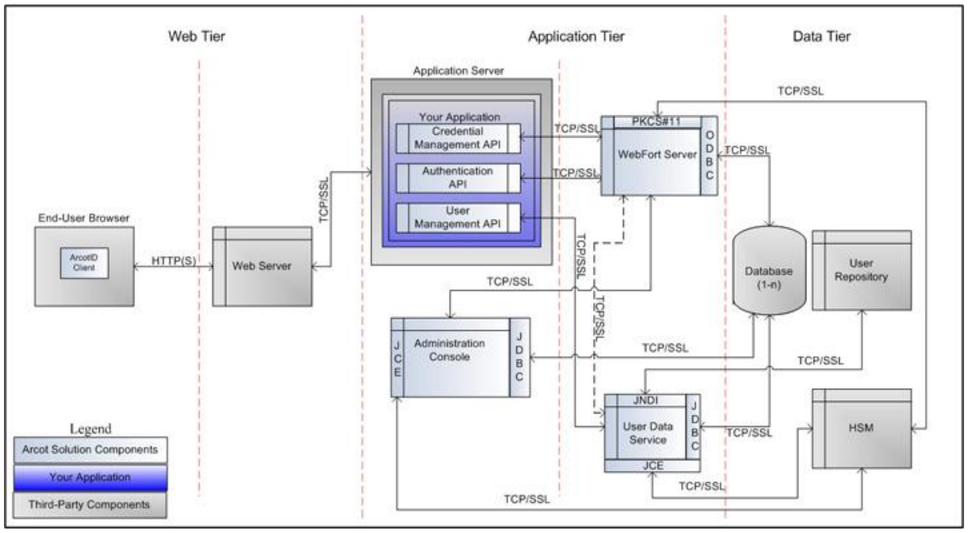
4. Set the value of port.<*n*> parameter to the port number on which the Transaction Native protocol is listening. The default port number for Transaction Native protocol is 9742.

Book: Refer to appendix, "Configuration Files and Options" in the *CA AuthMinder Installation and Deployment Guide* for more information on the webfort.authentication.properties and webfort.issuance.properties files.

Setting up SSL

To ensure integrity and confidentiality of the data being exchanged during a session, AuthMinder supports Secure Socket Layer (SSL) communication between Java SDKs and AuthMinder Server. By default, the communication mode between all the components is through Transmission Control Protocol (TCP).

The following figure shows the communication that are supported between AuthMinder components:



Important! Before you enable SSL communication between Java SDKs and AuthMinder Server, you must obtain a digital certificate from a trusted Certificate Authority and expose your application over an HTTPS-enabled server port.

To set up one-way SSL between Java SDKs (Authentication and Issuance) and AuthMinder Server, you must first configure the Transaction Native protocol by using the **Protocol Management** page of the Administration Console and then configure the webfort.authentication.properties and webfort.issuance.properties files.

In case of two-way SSL, you must create the client store using the **Trusted Certificates Authorities** page, configure the client store using the **Protocol Management** (**Transaction Native**) page, configure the client certificates using the **WebFort Connectivity** (**Transaction Native**) page of the Administration Console, and then configure the webfort.authentication.properties and webfort.issuance.properties files.

Note: If you want to enable SSL between Administration Web Service and AuthMinder Server, then you need to follow the steps mentioned in this section.

The following subsections walk you through the detailed steps for configuring:

- One-Way SSL (see page 140)
- Two-Way SSL (see page 142)

Note: In this communication, your application integrated with the Java SDKs is the client and AuthMinder Server is the server.

One-Way SSL

To enable SSL communication mode between Java SDKs and AuthMinder Server:

1. Access the Administration Console in a Web browser.
2. Ensure that you are logged in as the MA.
3. Activate the **Services and Server Configurations** tab in the main menu.
4. Ensure that the **WebFort** tab in the submenu is active.
5. Under the **Instance Configurations** section, click the **Protocol Management** link to display the Protocol Configuration page.
6. Select the **Server Instance** for which you want to configure the protocols.
7. In the **List of Protocols** section, click the **Transaction Native** protocol link

   The page to configure the protocol appears.
8. Configure the following fields:
   - Ensure that the **Protocol Status** is **Enabled**.
   - In the **Transport** field, select **SSL (1-Way)**.
   - Select **Key in HSM** if you want to store the SSL key in HSM.
   - (*Only* if you selected **Key in HSM** in the preceding step) Click the **Browse** button adjacent to the **Certificate Chain (in PEM Format)** field to select the AuthMinder root certificate.
   - Click the **Browse** button adjacent to the **P12 File Containing Key Pair** field to select the AuthMinder root certificate.
   - Enter the password for the PKCS#12 store in the **P12 File Password** field.
9. Click the **Save** button.
10. Restart the AuthMinder Server instance.
11. Navigate to the following location:
    - **On Windows:**
      <*install_location*>\Arcot Systems\sdk\client\java\properties
    - **On UNIX-Based Platforms:**
      <*install_location*>/arcot/sdk/client/java/properties
12. Open the webfort.authentication.properties file in an editor window.

    a. Set the following parameters:
       - authentication.transport = SSL (By default, this parameter is set to TCP.)
       - authentication.serverCACertPEMPath = <*absolute_path_of_Root_Certificate_in_PEM_FORMAT*>

       For example, you can specify authentication.serverCACertPEMPath = <install_location>/certs/<ca_cert>.pem.

Book: Refer to appendix, "Configuration Files and Options" in the *CA AuthMinder Installation and Deployment Guide* for more information on the webfort.authentication.properties file.

   b. Save the changes and close the file.

13. Open the webfort.issuance.properties file in an editor window.

    a. Set the following parameters:

       ■ issuance.transport = SSL (By default, this parameter is set to TCP.)

       ■ issaunce.serverCACertPEMPath = <*absolute_path_of_Root_Certificate_in_PEM_FORMAT*>

       For example, you can specify issuance.serverCACertPEMPath = <install_location>/certs/<ca_cert>.pem.

       Book: Refer to appendix, "Configuration Files and Options" in the *CA AuthMinder Installation and Deployment Guide* for more information on the webfort.issuance.properties file.

    b. Save the changes and close the file.

14. Restart the application server where Java SDKs are deployed.

Two-Way SSL

To enable SSL communication mode between Java SDKs and AuthMinder Server:

1. Enable the application server where Java SDKs are deployed for SSL communication. Refer to your application server vendor documentation for more information on how to do this.
2. Access the Administration Console in a Web browser.
3. Log in to Administration Console as the MA.
4. Activate the **Services and Server Configurations** tab in the main menu.
5. Activate the **WebFort** tab in the submenu.
6. Under **Instance Configurations**, click the **Trusted Certificate Authorities** link to display the corresponding page.

   The Trusted Certificate Authorities page appears.
7. Set the following information:
   - In the **Name** field, enter the name for the SSL trust store.
   - Click the **Browse** button to select the root certificate of the application server where Java SDKs are deployed.
8. Click the **Save** button.
9. Under **Instance Configurations**, click the **Protocol Management** link to display the corresponding page.

   The Protocol Configuration page appears.
10. Select the **Server Instance** for which you want to configure the protocols.
11. In the **List of Protocols** section, click the **Transaction Native** link.

    The page to configure the protocol appears.
12. Configure the following fields:
    - Ensure that the protocol is enabled.
    - In the **Transport** field, select **SSL (2-Way)**.
    - Select **Key in HSM** if you want to store the SSL key in HSM.
    - (*Only* if you selected **Key in HSM** in the preceding step) Click the **Browse** button adjacent to the **Certificate Chain (in PEM Format)** field to select the AuthMinder root certificate.
    - Click the **Browse** button adjacent to the **P12 File Containing Key Pair** field to select the AuthMinder root certificate.
    - Enter the password for the PKCS#12 store in the **P12 File Password** field.
    - Select the **Client Store** that you created in Step 7.
13. Click the **Save** button.

14. Restart the AuthMinder Server instance.
15. Activate the **Services and Server Configurations** tab in the main menu.
16. Activate the **WebFort** tab in the submenu.
17. Under **System Configuration**, click the **WebFort Connectivity** link to display the corresponding page.

    The WebFort Connectivity page appears.
18. Set the following for the **Transaction Native** protocol:
    - Ensure that the **IP Address** and **Port** number of the AuthMinder Server is set appropriately.
    - In the **Transport** field, select **SSL**.
    - Click the **Browse** button adjacent to the **Server CA Certificate in PEM** field to select the AuthMinder root certificate.
    - Click the **Browse** button adjacent to the **Client Certificate-Key Pair in PKCS#12** field to select the PKCS#12 file that contains the root certificate of the application server where Java SDKs are deployed.
    - Enter the PKCS#12 file password in the **Client PKCS#12 Password** field.
19. Click the **Save** button.
20. Restart the AuthMinder Server instance.
21. Navigate to the following location:
    - **On Windows:**
      <*install_location*>\Arcot Systems\sdk\client\java\properties
    - **On UNIX-Based Platforms:**
      <*install_location*>/arcot/sdk/client/java/properties
22. Open the webfort.authentication.properties file in an editor window.
    a. Set the following parameters:
       - authentication.transport = SSL (By default, this parameter is set to TCP.)
       - authentication.serverCACertPEMPath =
         <*absolute_path_of_Root_Certificate_in_PEM_FORMAT*>

       For example, you can specify authentication.serverCACertPEMPath = <install_location>/certs/<ca_cert>.pem.

       Book: Refer to appendix, "Configuration Files and Options" in the *CA AuthMinder Installation and Deployment Guide* for more information on the webfort.authentication.properties file.

    b. Save the changes and close the file.
23. Open the webfort.issuance.properties file in an editor window.
    a. Set the following parameters:

  - issuance.transport = SSL (By default, this parameter is set to TCP.)
  - issaunce.serverCACertPEMPath = <*absolute_path_of_Root_Certificate_in_PEM_FORMAT*>

  For example, you can specify issuance.serverCACertPEMPath = <install_location>/certs/<ca_cert>.pem.

  Book: Refer to appendix, "Configuration Files and Options" in the *CA AuthMinder Installation and Deployment Guide* for more information on the webfort.issuance.properties file.

b. Save the changes and close the file.

24. Restart the application server where your Java SDKs are deployed.

25. Verify that the AuthMinder Server is enabled for SSL communication by performing the following steps:

a. Navigate to the following location:

  - **On Windows**:
  <*install_location*>\Arcot Systems\logs
  - **On UNIX-Based Platforms**:
  <*install_location*>/arcot/logs

b. Open the arcotwebfortstartup.log file in a text editor.

c. Search for the following section:
Listing : [Successful listeners(Type-Port-FD)]

d. In this section, you must find the following line:
Transaction-Native.............................. :
[SSL-9742-<*Internal_listener_identifier*>-[subject [<*cert_subject*>] issuer [<*cert_issuer*>] sn [<*cert_serial_number*>] device [<*device_name*>]]]

e. Close the file.

Appendix D: SDK Exceptions and Error Codes

This appendix lists all exceptions and error codes that are returned by the AuthMinder 7.1.01 SDKs. It covers the following topics:

- Exceptions (see page 145)
- Error Codes (see page 147)

Exceptions

AuthMinder exceptions have been categorized as:

- Common Exceptions (see page 145)
- Issuance Exceptions (see page 146)
- Authentication Exceptions (see page 147)

Common Exceptions

The com.arcot.webfort.common.api.exception package provides the exceptions that are returned by AuthMinder Server and SDKs. The following table lists the exceptions of this package:

Classes	**Exception Returned By**	**Description**
CredentialNotFoundException	Server	This exception is returned if the credential with which the user is trying to authenticate was not found.
InvalidParamException	Server	This exception is returned if any of the parameter used in the operation has invalid value.
InvalidSDKConfigurationException	SDK	This exception is returned if the configuration file, whose absolute path is provided as the API input for initializing the API cannot be read.
SDKAlreadyInitializedException	SDK	This exception is returned if the SDK is already initialized.

Classes	**Exception Returned By**	**Description**
SDKException	SDK	This exception is the base class for all client-side exceptions.
SDKInternalErrorException	SDK	This exception occurs if: ■ The request is not valid. ■ The SDK failed to release connections. ■ The SDK generated an unclassified error.
SDKNotInitializedException	SDK	This exception is returned if you are using the function before initializing the SDK.
ServerException Server	Server	Base class for all server-side exceptions.
ServerUnreachableException SDK	SDK	This exception is returned if the SDK was not able to connect to the AuthMinder Server.
TransactionException	Server	This exception is returned if there is internal error while executing the transactions. For example, UDS is not running or the databases are down.
UserNotFoundException	Server	This exception is returned if the user trying to perform the operation is not enrolled in AuthMinder.

Issuance Exceptions

The com.arcot.webfort.issuance.api.exception package provides the exception classes that are returned based on user and credential status. The following table lists the issuance exceptions returned by AuthMinder Server:

Classes	**Description**
CredentialAlreadyExistsException	This exception is returned if you try to create the credential type that the user already has. The user cannot have multiple credentials of same type.

Classes	Description
UserAlreadyExistsException	This exception is returned if you try to create a user with a user name that already exists.

Authentication Exceptions

The com.arcot.webfort.authentication.api.exception package provides the exception classes that are returned based on user authentication and credential status. The following table lists the authentication exceptions returned by AuthMinder Server:

Classes	Description
AttemptsExhaustedException	This exception is returned if the user tried to authenticate with the wrong credential for the maximum allowed authentication attempts.
CredReissuedException	This exception is returned if the credential with which the user is trying to authenticate has been reissued.
InactiveAccountException	This exception is returned if the user trying to authenticate with the credential that is in one of the following states: ■ Disabled ■ Locked ■ Deleted ■ Verified (for OTP *only*)
InvalidCredException	This exception is returned if the credential provided by the user is not valid.

Error Codes

AuthMinder error codes have been categorized as:

- SDK Codes (see page 148)
- Server Codes (see page 150)

SDK Codes

The following table lists the SDK response codes, cause for failure, and solution wherever applicable:

SDK Response Code	Description	Possible Cause for Failure
0	The SDK has successfully sent the request and has received the response from the server or vice-versa. **Note:** This does not imply that the transaction was successful.	N/A
1	Internal error occurred in SDK due to some unexpected reason.	**Possible Cause:** Unexpected behavior by the SDK.
2 (Returned by SDKNotInitialized Exception)	SDK not initialized successfully.	**Possible Cause:** Returned when API is called without initializing. **Solution:** Check if the function to initialize the SDK has completed successfully.
3 (Returned by SDKAlreadyInitial izedException)	SDK is already initialized.	**Possible Cause:** User is trying to initialize the SDK that has already been initialized.
4	The configuration file, whose absolute path is provided as the input to the initialization API cannot be read.	**Possible Cause:** The configuration file path might be incorrect. **Solution:** Provide the correct configuration file path. **Possible Cause:** Permissions to read the configuration file are not set. **Solution:** Provide Read permission to the configuration file.

SDK Response Code	Description	Possible Cause for Failure
5 (Returned by ServerUnreachableException)	The SDK is not able to send requests to the configured server.	**Possible Cause:** Server host or port, or both might not be configured correctly. **Solution:** Provide correct host and port number. **Possible Cause:** Server might not be running. **Solution:** Start the server. **Possible Cause**: If SSL is configured, then certificates might not be configured correctly. **Solution:** Configure the SSL certificates correctly.
6	Buffer sent through the output structure is not sufficient.	**Possible Cause:** The buffer passed in the output structure(s) is not sufficient for the data to be filled. **Solution:** Send sufficient buffer to store all the data.

SDK Response Code	**Description**	**Possible Cause for Failure**
7 (Returned by InvalidSDKConfigurationException)	The SDK configuration is incorrect.	**Possible Cause:** Server host or port, or both might not be configured correctly. **Solution:** Provide correct host and port number. **Possible Cause**: If SSL is configured, then certificates might not be correct. **Solution:** Configure a valid client PKCS#12 file and server root CA certificate.
999 (Returned by SDKInternalErrorException)	Internal error.	**Possible Cause:** Unexpected SDK internal error.

Server Codes

The following table lists the response codes, reason codes, the cause for failure, and solution wherever applicable:

Response Code	**Reason Code**	**Description**	**Possible Cause for Failure**
0	0	Operation completed successfully.	N/A
	6100	Authentication succeeded, but the credential is in grace period.	**Action to Take:** Credential has already expired. Notify the user to get the credential reissued.
	6101	Authentication succeeded, but the credential is in warning period.	**Action to Take:** Credential is about to expire. Notify the user to get the credential reissued.
1000	0	Internal error.	**Possible Cause:** Unexpected internal error.

Respon se Code	Reason Code	Description	Possible Cause for Failure
	2000	Database is not operational.	**Possible Cause:** Database is not operational. **Solution:** Start the database. **Possible Cause:** Connection between the server and database is not complete. **Solution:** Establish the connection between server and database again using the database parameters available in arcotcommon.ini file.
1000	2001	Configuration is missing.	**Possible Cause:** Configuration required for processing the transaction is missing. **Solution:** Check the server transaction logs for details and ensure the required configuration is created and assigned. **Possible Cause:** Configuration required for processing the transaction is created but not available in server cache. **Solution:** Refresh server cache.
	2002	Transaction ID generation failed.	**Possible Cause:** Transaction ID generation failed due to internal error in the server. **Solution:** Most likely cause might be because of database failure. Check the server transaction logs for details and ensure appropriate action is taken based on the server logs.

Respon se Code	Reason Code	Description	Possible Cause for Failure
	6004	Internal error.	**Possible Cause:** Unexpected internal error.
1001	0	Access is denied.	**Possible Cause:** The operation being invoked is protected, and you need to authenticate. **Solution:** Obtain authentication credentials from your administration to include them in the call.
1050	0	Invalid parameter.	**Possible Cause:** The input parameter is invalid. **Solution:** Provide a valid parameter.
	2050	Value of one of the parameters used in the operation is empty.	**Possible Cause:** The parameter passed to the API is empty. **Solution:** Provide a non-empty value for the parameter. See appendix, "Input Data Validations" (see page 107) for the supported parameter values.
	2051	Length of one of the parameters used in the operation has exceeded the maximum allowed value. **Tip:** Length here refers to length of the parameter, for example password length.	**Possible Cause:** The length of the parameter passed to the API has exceeded the maximum value. **Solution:** Provide the parameter such that its length is less than or equal to the maximum allowed value. See appendix, "Input Data Validations" (see page 107) for the supported parameter values.

Respon se Code	Reason Code	Description	Possible Cause for Failure
	2052	Length of one of the parameters used in the operation is less than minimum allowed value.	**Possible Cause:** The length of the parameter passed to the API is less than minimum value. **Solution:** Provide the parameter such that the length of the parameter is greater than or equal to the minimum allowed value. See appendix, "Input Data Validations" (see page 107) for the supported parameter values.
1050	2053	Value of one of the parameters used in the operation exceeded the maximum allowed value. **Tip:** VALUE here refers to value of the parameter, for example ArcotID PKI Plain key length.	**Possible Cause:** The value of the parameter passed to the API has exceeded the maximum allowed value. **Solution:** Provide the parameter such that the value of the parameter is less than or equal to the maximum allowed value. See appendix, "Input Data Validations" (see page 107) for the supported parameter values.
	2054	Value of one of the parameters used in the operation is less than the minimum allowed value.	**Possible Cause:** The value of the parameter passed to the API is less than the minimum allowed value. **Solution:** Provide the parameter such that the value of the parameter is greater than or equal to the minimum allowed value. See appendix, "Input Data Validations" (see page 107) for the supported parameter values.

Respon se Code	Reason Code	Description	Possible Cause for Failure
	2055	Value of one of the parameters used in the operation is invalid.	**Possible Cause:** The value of the parameter passed to the API is invalid. For example, the allowed values for user status are 0 and 1. If you set the value of this as 5, then you will get this error. **Solution:** Provide valid value for the parameter. See appendix, "Input Data Validations" (see page 107) for the supported parameter values.
	2056	Value of one of the parameters used in the operation contains invalid characters.	**Possible Cause:** The parameter specified by ParameterKey contains invalid characters. **Solution:** Provide valid characters for the parameter that is specified by ParameterKey.
1050	2057	One of the parameters used in the operation does not meet the formatting requirements.	**Possible Cause:** The parameter specified by ParameterKey has invalid format. **Solution:** Provide valid format for the parameter that is specified by ParameterKey.
	2058	The password has less number of alphabets than the minimum allowed value.	**Possible Cause:** The password provided contains lesser number of alphabets than the password strength policy allows. **Solution:** Refer to the relevant password policy and ensure that the password strength is set correctly.

Respon se Code	Reason Code	Description	Possible Cause for Failure
	2059	The password has less number of numeric characters than the minimum allowed value.	**Possible Cause:** The password provided contains lesser number of numeric characters than the password strength policy allows. **Solution:** Refer to the relevant password policy and ensure that the password strength is set correctly.
	2060	The password has less number of ASCII special characters than the minimum allowed value.	**Possible Cause:** The password provided contains lesser number of ASCII special characters than the password strength policy allows. **Solution:** Refer to the relevant password policy and ensure that the password strength is set correctly.
	2061	Parameter is not supported for this operation.	**Possible Cause:** The parameter that is passed by the plug-in is not supported by the operation. For example, if you pass SAML token configuration name in the createCredential operation. **Solution:** Change the plug-in code appropriately.
1050	2063	Password is invalid.	**Possible Cause:** The PKCS#12 files are uploaded with a wrong password. **Solution:** Ensure that you use the correct password for the PKCS#12 files.

Respon se Code	Reason Code	Description	Possible Cause for Failure
	2064	Update operation is not supported for the parameter.	**Possible Cause:** You are trying to update a read-only parameter. **Solution:** None.
	2065	Parameter does not match.	**Possible Cause:** The organization name specified in the XML file to upload the OATH tokens does not match with organization name specified in the operation. **Solution:** Provide the correct organization name.
	6000	Duplicate questions are not supported.	**Possible Cause:** Two or more questions are same. **Solution:** Provide distinct questions.
	6001	Duplicate answers are not supported.	**Possible Cause:** Two or more answers are same. **Solution:** Provide distinct answers.
	6002	The question cannot be same as any of the answers.	**Possible Cause:** Question might be same as any of the answers. **Solution:** Provide distinct question and answer.

Respon se Code	Reason Code	Description	Possible Cause for Failure
1050	6007	Credential history check failed.	**Possible Cause:** The credential that you are trying to update failed the password history validation check. **Solution:** Ensure that the password that you have specified meets the history check criterion.
	6010	Question not found.	**Possible Cause:** Question that you are trying to update, delete, and for which you want to update answer does not exist. **Solution:** Ensure that you use the correct question.
	6105	Duplicate elements found.	**Possible Cause:** The PKCS12 file being uploaded in to the ArcotID PKI contains duplicate elements. **Solution:** Upload a PKCS#12 file that does not contain duplicate entries.
	6106	Invalid element reference.	**Possible Cause:** The element that you are trying to delete does not exist in the ArcotID PKI. **Solution:** Ensure that you use the correct element identifier.

Respon se Code	Reason Code	Description	Possible Cause for Failure
	6200	Event is already assigned.	**Possible Cause:** The event is already associated with an organization. **Solution:** Choose a different event to assign.
	6201	Duplicate events are not supported.	**Possible Cause:** The event list passed contains duplicate entries. **Solution:** Do not assign duplicate events.
1051	0	Invalid request.	**Possible Cause:** The packet received is invalid. **Solution:** 1. Ensure correct SDK is pointing to the server. 2.Ensure the port configured on the client-side refers to the appropriate server protocol.
1060	0	The request is noted.	**Possible Cause:** Caller verification of the QnA credential is successful. In this case server does not apply the authentication policy. **Solution:** NA
1100	0	Organization is not found.	**Possible Cause:**

Respon se Code	Reason Code	Description	Possible Cause for Failure
			Organization specified is not present. **Solution:** 1. Check if the organization with the given name is created. 2.After creating the organization, the server might need cache refresh. Refresh tt e server cache. 3. Check if the name of the organization passed is correct.
1101	0	Credential configuration not found for the organization.	**Possible Cause:** The configuration for the specified credential is not present. **Solution:** 1. Check if the configuration is created for this organization. 2. Check if the configuration is assigned to this organization. 3. Creating and assigning configuration might need cache refresh. Refresh the server cache.
1102	0	User not found.	**Possible Cause:** User is not present. **Solution:** Create the user or provide the user information correctly.
1103	0	Organization is not active.	**Possible Cause:** Organization is not active. **Solution:** Activate the organization using Administration Console.

Respon se Code	Reason Code	Description	Possible Cause for Failure
1104	0	Configuration already exists.	**Possible Cause:** The configuration that you are trying to create already exists. **Solution:** If you want to create a configuration, the use a different configuration name. If you want to update an existing configuration, then use the correct operation.
1150	0	User status is not active.	**Possible Cause**: User status is not active. **Solution:** Activate the user by using Administration Console.
5501	0	Data not found.	**Possible Cause:** There was no data found for the specified OATH token search criteria. **Solution:** Use a different search criteria.
5600	0	The RADIUS client IP is not valid.	**Possible Cause:** Client IP used in the RADIUS configuration is not valid. **Solution:** Ensure that you use an appropriate octet IP format.

Response Code	Reason Code	Description	Possible Cause for Failure
5601	0	The credential configuration is not valid.	**Possible Cause:** The configuration passed in the input is not valid. **Solution:** Based on the operation being performed there could be multiple reasons for this error. Check the parameter details in the response or check the server logs for further details.
	2003	Configuration organization does not match with the request organization.	**Possible Cause:** The organization name specified in the OATH token does not match with the organization name that you have specified in the operation. **Solution:** Ensure that you provide the correct organization name.
5601	6005	OATH token not found.	**Possible Cause:** OATH token being assigned is not uploaded to the organization or it might not be uploaded for the organization the current user belongs to. **Solution:** Check the token identifier and ensure that you upload the OATH token at the global level or for the current organization.
	6006	OATH token is already assigned to a user.	**Possible Cause:** The OATH token has already been assigned. **Solution:** Assign a different OATH token for the user.

Respon se Code	Reason Code	Description	Possible Cause for Failure
	6009	OATH token is abandoned.	**Possible Cause:** The OATH token has been used and abandoned. **Solution:** Assign a different OATH token for the user or reuse the same token by force-assigning the token.
	6104	Credential key is not active.	**Possible Cause:** The key with which the credential is protected is no longer ACTIVE. **Solution:** Reissue and use the new credential.
5602	0	The protocol is not valid.	**Possible Cause:** The protocol that you are trying to update or fetch is not valid. **Solution:** Use a valid protocol identifier.
5603	0	The credential configuration for the organization is not valid.	**Possible Cause:** The credential configuration name is not valid. **Solution:** You must provide a valid configuration name.
5605	0	SSL trust store group name is invalid.	**Possible Cause:** The provided organization name is not valid. **Solution:** You must provide a valid organization name.

Respon se Code	**Reason Code**	**Description**	**Possible Cause for Failure**
5606	0	SSL trust store group is invalid.	**Possible Cause:** SSL trust store with this name already exists. **Solution:** Create a trust store with a different name.
5607	0	Invalid WebFort Server instance name.	**Possible Cause:** Server instance name being set is not valid. **Solution:** Provide a valid instance name.
5608	0	A RADIUS client with the specified IP address is already configured.	Possible Cause: The IP address specified in the operation has already been configured. Solution: If the existing configuration is not correct, then delete that configuration and create a new configuration.
5700	0	Number of authentication attempts exceeded.	**Possible Cause**: Number of authentication attempts for the credential exceeded the allowed limit. **Solution:** The administrator must change the status of the credential from locked to active.
5701	0	Authentication token has expired.	**Possible Cause**: Authentication token submitted by the user is expired. **Solution:** Authenticate again.

Response Code	Reason Code	Description	Possible Cause for Failure
5702	0	Challenge has expired.	**Possible Cause:** Challenge is expired. **Solution:** Request for the challenge again.
5704	0	Credential has expired.	**Possible Cause:** The credential, which is provided by the user is expired. **Solution:** Get the new credential.
	0	The credential configured for ASSP has expired.	**Possible Cause:** The credential, which is provided by the user is expired. **Solution:** Get the new credential.
	6102	The credential validity period has not yet started.	**Possible Cause:** The credential has been created for future use. **Solution:** Use the credential that is within the validity period.
5705	0	Credential is not active.	**Possible Cause:** The credential, which is provided by the user is not active. **Solution:** The administrator must activate the credential.

Response Code	Reason Code	Description	Possible Cause for Failure
	0	Credential is not active" "ASSP" "The user ($$(USER)$$) account is inactive."	**Possible Cause:** The credential, which is provided by the user is not active. **Solution:** The administrator must activate the credential.
5706	0	Credential is reissued.	**Possible Cause:** Credential is reissued.
5707	0	The authentication credentials provided are incorrect.	**Possible Cause:** The credential details provided by the user are incorrect. **Solution:** Provide the credential details correctly.
	0	The ASSP authentication credentials provided are incorrect.	**Possible Cause:** The credential details provided by the user are incorrect. **Solution:** Provide the credential details correctly.
	6103	The authentication credentials provided are incorrect. Re-synchronize the credential.	**Possible Cause:** The OTP that is provided is not in the configured authentication window, but can be synchronized. **Solution:** Synchronize the OTP credential.

Respon se Code	Reason Code	Description	Possible Cause for Failure
5800	0	Credential not found for the user.	**Possible Cause:** The credential does not exist for the user. **Solution:** Create the credential. **Possible Cause:** The details provided by the user might be incorrect. **Solution:** Provide the correct details.
	0	ASSP credential not found for the user.	**Possible Cause:** The credential does not exist for the user. **Solution:** Create the credential. **Possible Cause:** The details provided by the user might be incorrect. **Solution:** Provide the correct details.
	6004	The credential not found for the user. It is already been deleted.	**Possible Cause:** The credential has already deleted. **Solution:** You can perform a fetch operation on the credential to understand the credential state. Reissue the credential, if required.
5801	0	Credential already present for the user.	**Possible Cause:** Credential already exists for the user.
	6008	Credential already present for the PAN.	**Possible Cause:** Credential already exists for the user.

Respon se Code	Reason Code	Description	Possible Cause for Failure
6500	0	The event is not supported.	**Possible Cause:** The event that being assigned to the plug-in is not supported by AuthMinder. **Solution:** Ensure that you use the supported events.
6501	0	The operation is not supported.	**Possible Cause:** The credential input provided is not valid. For example, you might have provided QnA input for the downloadCredential operation. **Solution:** Ensure that the input data that you provide is correct.

Appendix E: AuthMinder Sample Application

AuthMinder is shipped with a Sample Application, which demonstrates how to use the Java APIs.

Important! Sample Application must *only* be used as a code-reference and not for production.

Before you use the Sample Application, you must first configure it to communicate with AuthMinder Server. The following topics are covered in this appendix:

Note: This appendix *only* explains how to configure the Sample Application and select the ArcotID PKI Client type, but does not explain the other operations that can be performed easily using Sample Application.

Configuring Sample Application

The **Setup** page enables you to set the AuthMinder server host name or IP address, port at which the authentication and issuance service is available, and the Sample Application log file name and location. Perform the following steps to do so:

1. Launch Sample Application in a Web browser window. The default URL for Sample Application is:

   http://<host>:CA Portal/webfort-7.1.01-sample-application

   The WebFort 7.1.01 Sample Application page appears.

2. In the left-hand pane, expand the **Setup** button and then click **Server Connectivity** link to display the WebFort Server Connectivity page.

3. Specify the values for the configuration parameters listed in the following table:

Field	**Default Value**	**Description**
IP Address	localhost	The host name or the IP address where the AuthMinder Server is available.
Port	9742	The port at which the Authentication or the Issuance service is available.
Maximum Active Connections	64	The maximum number of connections maintained between the Sample Application and AuthMinder Server.

4. Click **Set Up** to configure the connection.

To configure the Sample Application to communicate with an additional AuthMinder Server instance:

1. Click the **[+]** sign preceding **Additional Server Configurations**.
2. Specify the **IP Address** and **Port** connection parameters.
3. Click **Set Up** to configure the connection.

Selecting ArcotID PKI Client

Before you perform any ArcotID PKI-related operations, you must choose the appropriate type of ArcotID PKI Client that you want to use, along with the required storage medium where you want to store the downloaded ArcotID PKI.

Note: The ArcotID PKI Client type and the download type that you select on this page will persist for your current browser session.

To select the ArcotID PKI Client:

1. Access the Sample Application using the following URL:

   http://<host>:CA Portal/webfort-7.1.01-sample-application/

2. In the left-hand pane, click **Setup** -> **ArcotID Client** to open ArcotID Client Settings page.
3. In the **Choose ArcotID Client** section, select the type of client that has to be used to authenticate the ArcotID PKI.

   Note: If you choose the Flash client, then the Sample Application must be enabled for HTTPS.

4. Select the type of medium where you want to store the ArcotID PKI from the **Choose ArcotID Download Type** section.

   See section, "Downloading ArcotID PKI (see page 36) for more information on the supported download types.

5. In the **Choose Where & When to Obtain the ArcotID Challenge** section, select the mode of obtaining the ArcotID PKI challenge.
6. Click **Select** to save the settings.
7. The "The operation was successful" message appears if the ArcotID PKI Client configuration was performed successfully.

Configuring Sample Application Log File

To configure the log file, which Sample Application uses to write the logs:

1. From the sidebar, expand the **Setup** button and then click **Logger** link to display the Logger Configuration page.
2. Enter the absolute path to the Sample Application log file in the **Log File Path** folder. By default, the Sample Application log file is generated in the *<APP_SERVER_HOME>* folder.
3. Select the **Log Level**. See the "Supported Severity Levels (see page 132)" section for more information on the log levels.
4. Click **Set Up** to configure the log file.

Creating Users

Note: From this release onwards, the user creation must be performed either using Administration Console or Web services.

To create users using Administration Console:

1. Log in to the Administration Console as a Global Administrator (GA) or an Organization Administrator (OA). The URL for the purpose is:

   http://<host>:CA Portal/arcotadmin/adminlogin.htm

2. If already not activated, activate the **Manage Users and Administrators** sub-tab under the **Users and Administrators** tab.
3. In the left-hand pane, under **Manage Users and Administrators**, click **Create User** to open the Create User page.
4. On the Create User page:
   a. Enter a unique user name, their organization name, and optionally, other user information in the **User Details** section
   b. If required, specify other user information in the corresponding fields on the page.
   c. Select the required **User Status**.
   d. Click **Create User.**

   The "Successfully created the user" message appears if the specified user was successfully added to the database.
5. Return to the WebFort Sample Application page.

Creating ArcotID PKI Credential

To create ArcotID PKI credential for users:

1. Access the Sample Application using the following URL:

   http://<host>:CA Portal/webfort-7.1.01-sample-application/

2. In the left-hand pane, click **ArcotID** -> **Issuance** -> **Create** to open Create ArcotID page.
3. Specify the name of the user you created in the **User Name** field.
4. If required, specify the user's organization in the **Organization** field.
5. Specify the password to be used for authentication in the **ArcotID Password** field.
6. If required, specify the profile that has to be used to issue ArcotID PKI in the **Profile Name** field.
7. If required, specify the name-vale pairs of the **Unsigned Attributes**. The attributes are set in the unsigned portion of the ArcotID PKI.
8. If required, specify the **Custom Attributes** to be used for creating the ArcotID PKI.
9. If required, specify the **Additional Input** that you want to pass to the AuthMinder Server.
10. If required, pass the following **Transaction Logging Parameters**:
    - In the **Log Level** field, choose the logging level.

      See "Supported Severity Levels" (see page 132) for more information.
    - Select **Enable Trace Logging** if you want to capture flow details.
    - Select **Enable DB Logging** if you want to log the database activities.
    - Select **Enable Sensitive Data Logging** if you want to log the sensitive data.
11. Click **Create** to create the credential.

    The "The operation was successful" message appears if the ArcotID PKI was created successfully for the user.

Downloading ArcotID PKI

To download the ArcotID PKI:

1. Access the Sample Application using the following URL:

   http://<host>:CA Portal/webfort-7.1.01-sample-application/

2. In the left-hand pane, click **ArcotID** -> **Issuance** -> **Download** to open the Download ArcotID page.
3. Specify the name of the user you created in the **User Name** field.
4. If required, specify the user's organization in the **Organization** field.
5. If required, specify the profile that has been used to issue ArcotID PKI in the **Profile Name** field.
6. If required, specify the **Additional Input** that you want to pass to the AuthMinder Server.
7. If required, pass the following **Transaction Logging Parameters**:
   - In the **Log Level** field, choose the logging level.

     See "Supported Severity Levels" (see page 132) for more information.
   - Select **Enable Trace Logging** if you want to capture flow details.
   - Select **Enable DB Logging** if you want to log the database activities.
   - Select **Enable Sensitive Data Logging** if you want to log the sensitive data.
8. Click **Download** to download the user's ArcotID PKI.

Authenticating Using ArcotID PKI

To authenticate using the ArcotID PKI:

1. Access the Sample Application using the following URL:

   http://<host>:CA Portal/webfort-7.1.01-sample-application/

2. In left-hand pane, click **ArcotID** -> **Authentication** -> **Authenticate** to open the ArcotID Authentication page.
3. Specify the name of the user you created in the **User Name** field.
4. Specify the user's organization in the **Organization** field.
5. Specify the user's ArcotID PKI password in the **ArcotID Password** field.
6. If you are using aliases to identify the users, then specify the **Application Context** based on the alias of the user that you want to authenticate.
7. If required, select the **Token Type** that has to be returned to the user after successful authentication.

   See "Verifying the Authentication Tokens" (see page 102) for more information on token types.

8. If required, specify the **Authentication Policy Name** that has to be used for authenticating users.
9. If you have selected SAML as the token type, then specify the **SAML Policy Name** to be used.
10. If required, specify the **Additional Input** that you want to pass to the AuthMinder Server.
11. If required, pass the following **Transaction Logging Parameters**:
    - In the **Log Level** field, choose the logging level.

      See "Supported Severity Levels" (see page 132) for more information.

    - Select **Enable Trace Logging** if you want to capture flow details.
    - Select **Enable DB Logging** if you want to log the database activities.
    - Select **Enable Sensitive Data Logging** if you want to log the sensitive data.
12. Click **Authenticate** to verify the user's ArcotID PKI.