Arcot WebFort VAS®
Java Developer's Guide
Version 6.0

ARCOT™

Arcot WebFort VAS Java Developer's Guide
Version 6.0
August 2009
Part Number: AWF01-004DC-06000

Trademarks

Patents

This software is protected by United States Patent No. 6,170,058, 6,209,102 and other patents pending.

Arcot Systems, Inc., 455 West Maude Avenue, Sunnyvale, CA 94085

Third-Party Software

All third-party software used by Arcot WebFort and related components are listed in the appendix "Third-Party Software Licenses" in the *Arcot WebFort 6.0 Installation and Deployment Guide*.

Contents

Preface

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

See the following JSP files shipped as a part of the **Sample Application** to understand the WebFort APIs for:

- **User Operations**

    - `ArWFCreateUser.jsp` - This page is used to create the user.

    - `ArWFUpdateUser.jsp` - This page is used to update the user information.

    - `ArWFFetchUser.jsp` - This page is used to fetch the user details.

- **Credential Operations**

    - `ArWFCreate<Credential>.jsp` - This page is used to create credential.

    - `ArWF<Credential>Authenticate.jsp` - This page is used to authenticate the user with the credential.

    - `ArWFFetch<Credential>.jsp` - This page is used to fetch the credential details.

    - (Only ArcotID) `ArWFSelectArcotIDClient.jsp` - This page is used to select the ArcotID Client.

    - (Only ArcotID) `ArWFDownloadArcotID.jsp` - This page is used to download the ArcotID Client.

> **NOTE:** For the operations not covered by the Sample Application, see
> relevant Javadocs.

Intended Audience

This guide is targeted at Java application programmers who need to use the APIs and
functions provided with Arcot WebFort to implement issuance and authentication
features provided by WebFort.

Readers should be familiar with:

• Java programming

• Database architecture and concepts

• Security management concepts

• Authentication and authorization concepts

• Internet protocols, including HTTP, HTTPS, and TCP/IP

• Secure Sockets Layer (SSL) communications and the related concepts (public and
  private key exchange, digital certificates and signatures, and certificate authorities)

• Security Assertion Markup Language (SAML) basics

Information Included in this Guide

This guide is organized as follows:

• **Chapter 1, "Getting Started"**, introduces you to the WebFort Java SDK and walks
  you through the prerequisites before you start programming.

• **Chapter 2, "Understanding WebFort WorkFlows"**, covers the typical workflows
  that you can implement by using WebFort SDKs.

• **Chapter 3, "Before You Use the SDK"**, describes what API-specific JAR files and
  Properties files to include in CLASSPATH.

• **Chapter 4, "Performing Issuance Operations"**, describes how to create and manage
  users and their credentials by using the Issuance APIs.

• **Chapter 5, "Integrating ArcotID Client with Application"**, describes how to
  implement the required ArcotID type (Flash or Signed Java Applet) for ArcotID
  authentication.

- **Chapter 6, "Authenticating Users"**, describes how to authenticate users by using one of the authentication mechanisms provided by the Authentication API.

- **Chapter 7, "Using Custom APIs"**, describes how to create custom credential and authenticate using them.

- **Appendix A, "Input Data Validations"**, lists the criteria that is used to check the SDK input parameters.

- **Appendix B, "WebFort Logging"**, provides the details of different log files and supported log levels that you can use for debugging purposes.

- **Appendix C, "Additional Settings",** provides the details on how to set up multiple WebFort Server instances, and SSL between Java SDKs and WebFort Server.

- **Appendix D, "SDK Exceptions"**, lists all exceptions that are returned by the Issuance and Authentication SDKs.

- **Appendix E, "WebFort Error Codes"**, lists WebFort SDK and Server error codes.

- **Appendix F, "WebFort Sample Application"**, describes the WebFort workflows demonstrated by the Sample Application that is shipped with WebFort.

- **Appendix G, "Glossary"**, lists the key terms used in the guide.

Related Publications

Other related publications are as follows:

Arcot WebFort 6.0 Installation and Deployment Guide	This guide provides the information to install and configure WebFort.
Arcot WebFort 6.0 Administration Guide	This guide includes the information to administer and configure WebFort.
Arcot WebFort 6.0 Business Logic Extension Guide	This guide provides information about how to write a plug-in or Callout to extend the existing authentication and issuance processes.
ArcotID Client 6.0 Reference Guide	This guide describes ArcotID Client types provided by the client.
ArcotID Flash Client 6.0 API Guide	This guide explains the ArcotID Flash client APIs provided by the client.

Conventions Used in This Book

The following typographical conventions are used in this guide.

Type	Usage	Example
Bold	Screen Items	Click the **Install** button to install the product.
Italic	Key Words	First time log in to the *Administration Console* must be done using Master Admin credentials.
	Names of Publications	For more information, see the *Arcot WebFort 6.0 Administration Guide*.
	Emphasis	*Never* give anyone your password.
Cross reference	Links in the guide	Refer to the section **Deployment Overview** for more information.
Fixed-width	Command-line input or output	`# cd /opt/arcot`
	Code Samples	`var walletname = "GuestUser";`

	Text File Content	`[arcot/db/primarydb]`
		`# The name of the data source as`
		`# defined in ODBC.`
		`Datasource.1=ArcotWebFortDatabase`

	File names	`arcotcommon.ini`

Contacting Support

If you need help, contact Arcot Support at:

Email	support@arcot.com
Web site	http://support.arcot.com

Chapter 1

Getting Started

This chapter discusses the APIs provided by WebFort Java SDK and the checks that you must perform before using the Java SDK:

- **Introduction to the WebFort Java SDK**

- **WebFort SDK Features**

- **Before You Begin**

Introduction to the WebFort Java SDK

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

Issuance SDK

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

- Create users

- Create credentials for the users

- Fetch user information

- Update user information

- Perform credential lifecycle management operations, such as enabling, disabling, resetting, resetting validity, and deleting.

WebFort 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 WebFort Server by using Secure Socket Layer (SSL). To set up SSL between SDK and WebFort Server, you must edit the WebFort properties files, see **"Setting up SSL"** for more information on how to do this.

- **Failover**

  Java SDKs support failover mechanism, if an instance of WebFort Server is not operational, then the SDKs automatically connect to any of the additional configured instances. See **"Configuring Multiple WebFort Server Instances"** 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"** or **"Initializing the Authentication SDK"** 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, 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.

- **Support for Custom APIs**

  The Custom APIs enables you to support additional authentication methods by parallely supporting any of the WebFort native authentication methods. See **Chapter 7, "Using Custom APIs"** for more information on Custom APIs.

Before You Begin

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

*   WebFort VAS® is installed and running on the required operating system.

    Refer to *Arcot WebFort 6.0 Installation and Deployment Guide* for the installation and configuration details.

*   You have installed the correct version of the JDK required for using the WebFort Java SDK.

    See *Arcot WebFort 6.0 Installation and Deployment Guide* for this information.

Chapter 2

Understanding WebFort WorkFlows

WebFort 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. Arcot does not mandate you to follow the exact steps for each procedure mentioned in this chapter.

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

- **Migrating Existing Users**

- **ArcotID Authentication Workflow**

- **ArcotID Roaming Download Workflow**

- **Forgot Your Password Workflow**

- **Workflow Summary**

Migrating Existing Users

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

> **NOTE:** If you are using a Directory Service (LDAP), then WebFort must be connected to the LDAP and you must map the LDAP attributes to the attributes supported by WebFort. See *Arcot WebFort 6.0 Administration Guide* for more information on this.

- **Migrating All Users**

- **Migrating Selected Users**

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 or you can set the existing password as the ArcotID 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 `ArcotIDIssuance` class.**

   Your application invokes the create() method in the ArcotIDIssuance class to create ArcotID for the user.

4. **WebFort returns the result.**

   WebFort posts the result of the create operation to the application.

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

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

Figure 2-1 illustrates the workflow for migrating individual user.

Figure 2-1   Migrating All Users
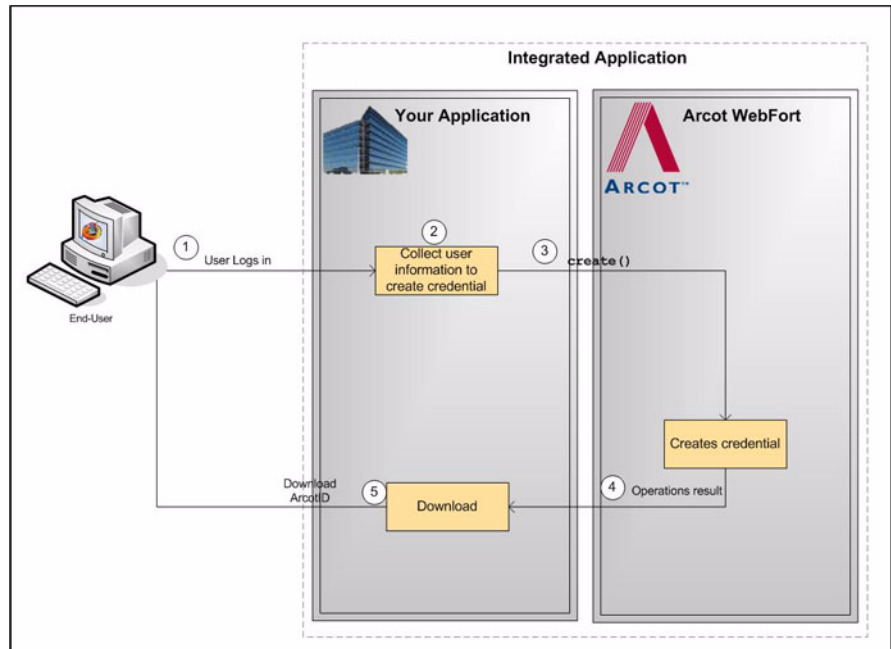
Migrating Selected Users

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 or you can set the existing password as the ArcotID password, and collect questions and answers if QnA is used for secondary authentication.

5. **Your application invokes the `create()` method in the `ArcotIDIssuance` class.**

   Your application invokes the create() method in the ArcotIDIssuance class to create ArcotID for the user.

6. **WebFort returns the result.**

   WebFort posts the result of the create operation to the application.

7. **Application downloads the ArcotID on the user's system.**

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

Figure 2-2 illustrates the workflow for migrating the users to ArcotID authentication in bulk.

Figure 2-2   Migrating Selected Users
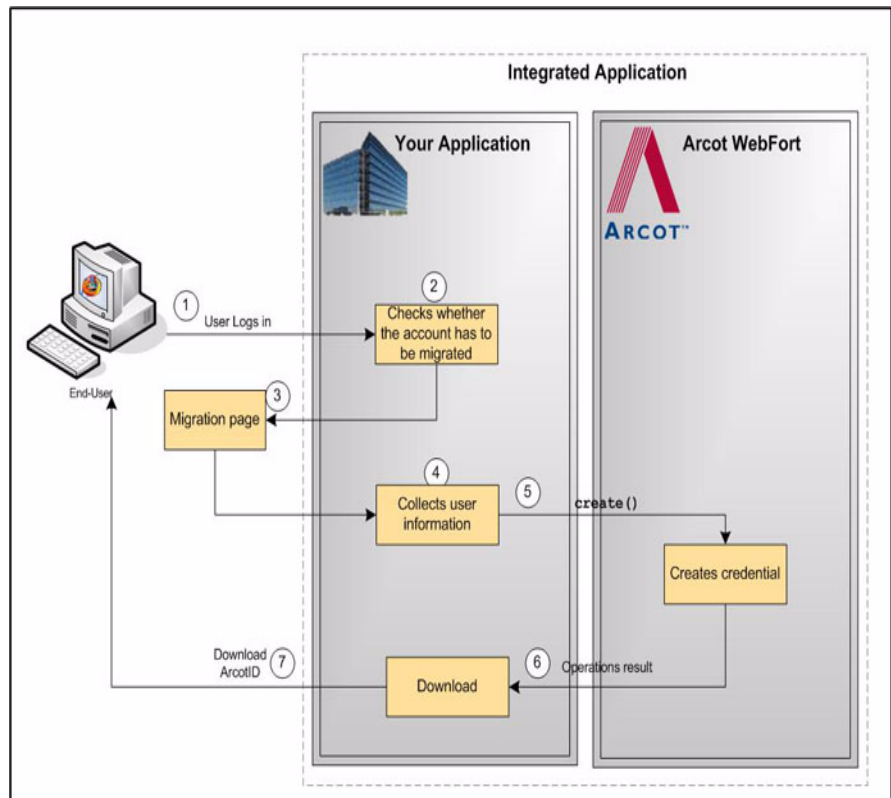
ArcotID Authentication Workflow

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

Username-password, QnA, and OTP are simple password-based authentication methods, while ArcotID is a challenge-response type of authentication. The following workflow lists the steps for **ArcotID authentication**:

> **NOTE:** In case of other credentials, refer to **Chapter 6, "Authenticating Users"** for details of methods to invoke.

1. **Application calls the WebFort's `ArcotIDAuth.getChallenge()` function**.

   Your application loads the ArcotID Client and makes an explicit call to the `getChallenge()` function in `ArcotIDAuth` interface. See **"ArcotID Authentication" on page 72** for more information on the API.

2. **User provides the credentials.**

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

3. **Your application invokes the ArcotID Client.**

   The ArcotID Client signs the challenge.

4. **WebFort verifies the signed challenge.**

   Your application invokes the `verifySignedChallenge()` function to verify the challenge that is signed by using the ArcotID password.

5. **WebFort authenticates the user.**

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

   See **"Initializing the Authentication SDK"** for more information on the different tokens supported by WebFort.

Figure 2-3 illustrates the workflow for ArcotID authentication process.

Figure 2-3   ArcotID Authentication
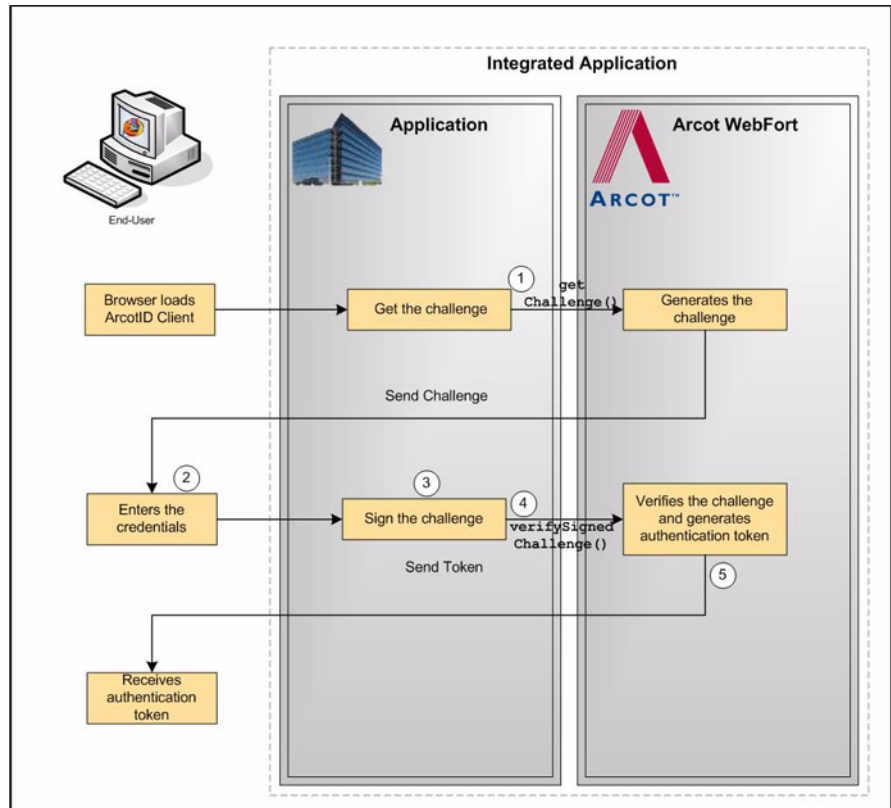
ArcotID Roaming Download Workflow

To perform ArcotID authentication, the ArcotID 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 primary system, where their ArcotID is stored, then the user has to download the ArcotID from the WebFort Server and then perform the authentication. This process of downloading the ArcotID to different system is called *Roaming Download*.

The typical steps for roaming download of the ArcotID are:

1. **User logs into your online application.**

   Your application authenticates the user.

2. **User chooses to download the ArcotID.**

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

3. **WebFort 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 WebFort's `ArcotIDAuth.getArcotID()` function.**

   If the secondary authentication was successful, only then your application should call the `getArcotID()` function in the `ArcotIDAuth` interface. This call downloads the corresponding ArcotID (encoded in the base-64 format) to the application.

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

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

Figure 2-4 illustrates the workflow for roaming download of ArcotID.

Figure 2-4   Roaming Download of ArcotID
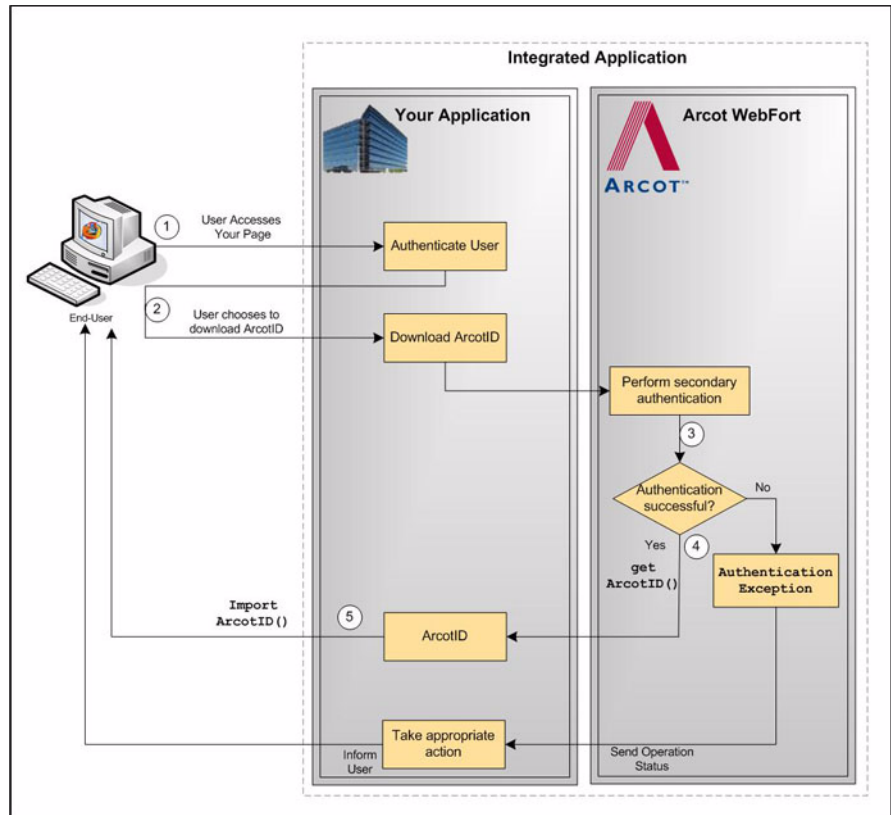
Forgot Your Password Workflow

If a user forgets their ArcotID 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. **WebFort 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 WebFort'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 Password" for more information on the APIs used to reset the credential.

Figure 2-5 illustrates the Forgot Your Password workflow.

Figure 2-5 FYP WorkFlow
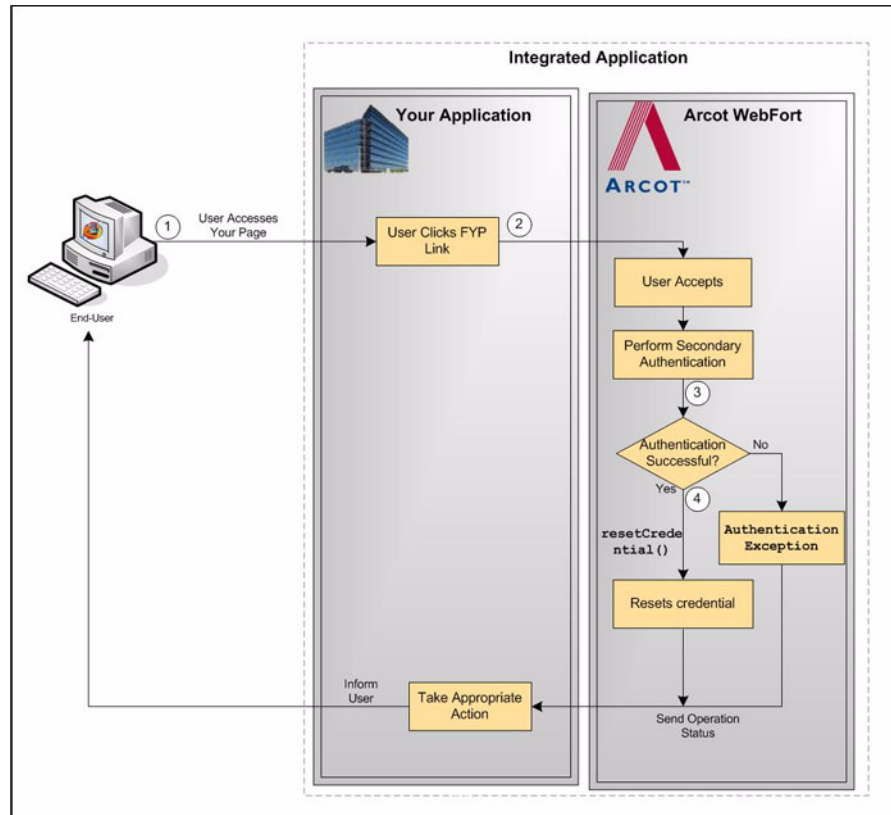
Workflow Summary

Table 2-1 provides a brief summary of the workflows that can be implemented by using the WebFort APIs.

Table 2-1   Summary of WebFort Workflows

Workflow	Description	Dependant Workflows
Enrollment	Creates a new user in the WebFort database, when you call `create()` function in the `UserIssuance()` class.	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
ArcotID Download	Downloads the ArcotID of the user to the system.	• Enrollment • Creating the Credentials
Migration	Migrates the user to ArcotID 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 WebFort. The WebFort 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 WebFort demonstrates the usage of the Java classes and methods. See **Appendix F, "WebFort Sample Application"** for more information on WebFort 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 WebFort SDK Javadocs**

- **Adding Authentication Files in CLASSPATH**

- **Adding Issuance Files in CLASSPATH**

Accessing WebFort SDK Javadocs

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

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

The Authentication SDK Javadoc
(`Arcot-WebFort-6.0-authentication-sdk-javadocs.zip`) and the Issuance SDK Javadoc (`Arcot-WebFort-6.0-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 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\java\lib`

For Unix Platforms: `<install_location>/arcot/sdk/java/lib`

Table 3-1   Authentication SDK JARs

File Name	Description
`arcot/arcot-pool.jar`	Used to implement connection pooling using commons pooling.
`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.
`external/bcprov-jdk14-139.jar`	Used for cryptographic functions. For example, reading server PEM certificate.
`external/commons-httpclient-3.1.jar`	Used for HTTP-based communication.
`external/commons-lang-2.0.jar`	Contains the utilities for string and exception handling that are used by Authentication SDK.
`external/commons-pool-1.4.jar`	Used for connection pooling.
`external/log4j-1.2.14.jar`	The Apache package for controlling run-time logging behavior of applications.

Properties File

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

> **NOTE:** You can also copy the WebFort 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\java\properties

For Unix Platforms: `<install_location>`/arcot/sdk/java/properties

> **NOTE:** If you want to edit the properties file to configure more WebFort Server instances and for SSL, then refer to **Appendix C, "Additional Settings"** for more information.

Adding Issuance Files in CLASSPATH

To use the Issuance APIs, you must add its JAR files 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\java\lib`

For Unix Platforms: `<install_location>/arcot/sdk/java/lib`

Table 3-2   Issuance SDK JARs

File Name	Description
`arcot/arcot-pool.jar`	Used to implement connection pooling using commons pooling.
`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/activation-1.1.jar`	Used by Axis JAR files.
`external/axiom-api-1.2.7.jar`	
`external/axiom-impl-1.2.7.jar`	
`external/axis2-adb-1.4.jar`	
`external/axis2-java2wsdl-1.4.jar`	
`external/axis2-kernel-1.4.jar`	
`external/backport-util-concurrent-2.2.jar`	
`external/bcprov-jdk14-139.jar`	
`external/commons-codec-1.3.jar`	
`external/commons-collections-3.1.jar`	
`external/commons-httpclient-3.1.jar`	

Table 3-2   Issuance SDK JARs

File Name	Description
`external/commons-lang-2.0.jar`	Used by AXIS JAR files.
`external/commons-logging-1.1.jar`	
`external/commons-pool-1.4.jar`	
`external/geronimo-spec-jms-1.1-rc4.jar`	
`external/log4j-1.2.14.jar`	
`external/neethi-2.0.jar`	
`external/stax-api-1.0.1.jar`	
`external/wsdl4j-1.6.2.jar`	
`external/wstx-asl-3.2.0.jar`	
`external/XmlSchema-1.2.jar`	

Properties File

The `webfort.issuance.properties` file containing the WebFort 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"**), which accepts the `name-value` pairs as input parameters.

> **NOTE:** You can also copy the WebFort 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\java\properties`

For Unix Platforms: `<install_location>/arcot/sdk/java/properties`

> **NOTE:** If you want to edit the properties file to configure more WebFort Server instances and for SSL, then refer to **Appendix C, "Additional Settings"** for more information.

Chapter 4

Performing Issuance Operations

For WebFort to authenticate users, an account for each user has to be created in the database, which is a one-time process. The user can either be created in the Arcot database or WebFort can be configured to connect to LDAP for user information.

> **IMPORTANT:** If your WebFort deployment is accessing the user information from the LDAP, then you can *only* use the APIs that reads the user details.

This chapter provides description of the APIs that are used for user and credential operations that include:

• **Initializing the Issuance SDK**

• **User Operations**

• **Credential Operations**

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.

Table 4-1   SDK Initialization by Using Map

Description	Input Values	Output Value

Initializes the Issuance SDK by using the provided map.	• map
	  The key-value pair specifying the configuration information. The supported keys are:
	
	  1. issuance.host.1
	     The IP address of the system where WebFort Server is available.
	
	  2. issuance.port.1
	     The port on which the Transaction Web Services protocol is listening. Default value is 9744.
	
	• 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 parameters (transport, host, and port) in the
webfort.issuance.properties file are same as that of map. The following table
provides the details of the **init()** method.

Table 4-2   SDK Initialization by Using Properties File

Description	Input Values	Output Value
Initializes the Issuance SDK by using the properties file.	• location The absolute path of the 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.

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.

Table 4-3   Releasing the API

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

User Operations

This section covers the following user operations:

- **Preparing the User Input**
- **Preparing Additional Input**
- **Creating Users**
- **Disabling User Accounts**
- **Enabling the User**
- **Reading User Details**
- **Updating User Details**
- **User Operations Summary**

Preparing the User Input

All issuance requests that are presented to the WebFort Server must be prepared, so that they are in a format that WebFort Server can process. To prepare the data prior to any user operations, you must use **UserInput** class. This class enables you to set the information used during user operations.

Depending on the operation that you are performing, input data preparation might include:

- **Setting the First Name**
- **Setting the Last Name**
- **Setting the Middle Name**
- **Setting the Email ID**
- **Setting the Telephone Number**
- **Setting PAM**
- **Adding Custom Attributes**

Setting the First Name

To set the first name of the user, invoke the **setFirstName()** method of the UserInput class.

Setting the Last Name

To set the last name of the user, invoke the **`setLastName()`** method of the `UserInput` class.

Setting the Middle Name

To set the middle name of the user, invoke the **`setMiddleName()`** method of the `UserInput` class.

Setting the Email ID

To set the email ID of the user, invoke the **`setEmailId()`** method of the `UserInput` class.

Setting the Telephone Number

To set the telephone number of the user, invoke the **`setTelephoneNumber()`** method of the `UserInput` class.

Setting PAM

To set the Personal Assurance Message (PAM) of the user in the WebFort database, invoke the **`setPAM()`** method of the `UserInput` class.

The PAM verifies the server to the client, and is displayed when the user tries to access the WebFort-protected resource.

Adding Custom Attributes

The **`setCustomAttribute()`** method is used to define any custom attribute for the user.

These attributes are used in addition to the standard attributes provided by the `UserInput` class.

> **NOTE:** WebFort Server validates all input data that you send to it. See
> **Appendix A, "Input Data Validations"** for more information on the
> criteria that the WebFort Server uses to validate this input data.

Preparing Additional Input

You need to prepare additional inputs if:

- You plan to augment WebFort's standard user issuance capability by implementing callouts or plug-ins.

- You plan to write completely new custom user issuance methods.

In all of these cases, you need to set the extra information that must be sent to WebFort Server in name-value pairs. WebFort's com.arcot.webfort.common.api package 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 include:

- **AR_WF_LOCALE_ID**

  Specifies the locale that WebFort will use while returning the messages back to your calling application.

- **AR_WF_CALLER_ID**

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

Creating Users

To create a user in WebFort database, you need to use the **UserIssuance** interface, as follows:

1. Use the **UserInput** class to obtain the methods that set the information of the user.

2. (**Optional**) If you are implementing a callout, plug-in, or any other custom method for issuance operations, 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" for more information.

3. Invoke the **create()** method of the UserIssuance interface to create the user account.

   This method returns an instance of the **UserResponse** interface, which specifies the user and the transaction 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" and "Common Exceptions" 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 User Accounts

The user account can be disabled for a specified period. For example, if in an organization an employee goes for long vacation, then their account can be disabled to prevent any unauthorized access during the specified period. The user, whose account is disabled cannot perform any credential operations.

To disable a user account:

1. Use the **UserInput** class to obtain the methods that set the information of the user.

2. (**Optional**) If you are implementing a callout, plug-in, or any other custom method for issuance operations, 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"** for more information.

3. Invoke the **disable()** method of the UserIssuance interface to disable the user account.

    This method returns an instance of the **UserResponse** interface, which specifies the user and the transaction 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"** and **"Common Exceptions"** 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 the User

The **enable()** method is used to activate a user account that was disabled earlier.

> **NOTE:** You can only enable the user accounts that are disabled.

To enable a user:

1. Use the **UserInput** class to obtain the methods that set the information of the user.

2. (**Optional**) If you are implementing a callout, plug-in, or any other custom method for issuance operations, 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" for more information.

3. Invoke the **enable()** method of the UserIssuance interface to enable the user account.

   This method returns an instance of the **UserResponse** interface, which specifies user and the transaction 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" and "Common Exceptions" 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 User Details

To retrieve the details of a user from the WebFort database, you need to use the **UserIssuance** interface.

To read the user details:

1. Use the **UserInput** class to obtain the methods that set the information of the user.

2. (**Optional**) If you are implementing a callout, plug-in, or any other custom method for issuance operations, 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" for more information.

3.   Invoke the **fetch()** method of the UserIssuance interface to read the user
     details.

     This method returns an instance of the **UserResponse** interface, which specifies
     the user and the transaction 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" and "Common Exceptions" 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.

Updating User Details

The Issuance API provides **update()** method to change the user account details in the
database. For example, these details include user's email address or contact number.

> **NOTE:** You *cannot* change the user name or the name of the organization to
> which the user belongs.

To update a user's details:

1.   Use the **UserInput** class to obtain the methods that set the information of the
     user.

2.   (**Optional**) If you are implementing a callout, plug-in, or any other custom method
     for issuance operations, 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" for more information.

3.   Invoke the **update()** method of the UserIssuance interface to update the user
     details.

     This method returns an instance of the **UserResponse** interface, which specifies
     the user and the transaction 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" and "Common Exceptions" 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.

User Operations Summary

The following table provides a summary of the input parameters required for performing all user operations discussed in this chapter.

Table 4-4   User Operations

Operation	Input Required	Expected Output
Create	• User name (`userName`)	• `UserResponse`
Update	• (Optional) Organization name (`orgName`) **NOTE:** If the organization name is not provided, then the user is assumed to belong to default organization. • (Optional) Additional Input (`AdditionalInput`) • (Optional) Personal Assurance Message (`PAM`) • (Optional) User attributes (`firstName`, `lastName`, `middleName`, `telephoneNumber`, and `emailId`) • (Optional) Custom attributes defined	
Disable **Enable** **Fetch**	• User name (`userName`) • (Optional) Organization name (`orgName`) **NOTE:** If the organization name is not provided, then the user is assumed to belong to default organization. • (Optional) Additional Input (`AdditionalInput`)	

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 WebFort, and can be performed by using any of the following method:

- **By using WebFort SDKs**

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

- **By using WebFort 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.)

  Refer to *Arcot WebFort 6.0 Administration Guide* for more information on using the Administration Console.

This section covers the following credential lifecycle operations:

- **Preparing the Input**
- **Checking the User Status**
- **Creating Credentials**
- **Disabling Credentials**
- **Enabling Credentials**
- **Resetting Credential**
- **Fetching Credential Details**
- **Reissuing Credentials**
- **Resetting Credential Validity**
- **Resetting Credential Notes**
- **Deleting Credentials**
- **Setting Unsigned Attributes**
- **Deleting Unsigned Attributes**
- **Reading the Output**

- **User Operations Summary**

- **Credential Operations Summary**

> **NOTE:** Each operation discussed in this chapter can be performed
> simultaneously on 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, QnA, and OTP, and the
> ArcotID and OTP creation was successful, while the QnA creation failed,
> then all the three credentials have to be created again.

Preparing the Input

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

- **Common Input**

- **Credential-Specific Input**

- **Preparing Additional Input**

Common Input

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

- **Validity**

- **Notes**

- **Profile Name**

Validity

When creating the credential, the Issuance API enables you to set a period for which the
credential will be valid. Invoke the **setValidity()** method of the
CredentialInput class to set validity for the credential. This is taken as input by
create() or resetValidity() methods.

Notes

The Issuance API enables you to add notes 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 on more than five systems, then you can create a
note with this information. This is taken as input by create() or resetNotes()
methods.

To add notes, 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.

Credential-Specific Input

The **com.arcot.webfort.issuance.api** package provides the interface that you can use to set the credential-specific information.

Preparing ArcotID Input

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

- Unsigned Attributes

- Password

- ArcotID Attributes

Unsigned Attributes

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

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

To set unsigned attributes:

1. Use the **ArcotIDInput** class to obtain the methods that set the information of the ArcotID.

2. Use **ArcotIDAttribute** class to define the unsigned attributes to set in the ArcotID.

3. Invoke the **setUnsignedAttributes** method in the **ArcotIDInput** class.

Password

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

To set the ArcotID password:

1. Use the **ArcotIDInput** class to obtain the methods that set the information of the ArcotID.

2. Invoke the **setPassword** method in the **ArcotIDInput** class.

ArcotID Attributes

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

To fetch ArcotID attributes:

1. Use the **ArcotIDInput** class to obtain the methods that set the information of the ArcotID.

2. Invoke the **setFetchAttribute** method in the **ArcotIDInput** class.

Preparing QnA Input

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

To add the questions and answers:

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

2. Invoke the **setQuestionAnswer** method in the **QnAInput** class.

Preparing Username-Password Input

The password for the username-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 QnA.

2. Invoke the **setPassword** method in the **UPInput** class.

Checking the User Status

WebFort uses the user status information *before* performing some of the credential operations. A user's status in the database can be either ACTIVE or DISABLED.

For Issuance SDK to perform these checks, you must enable this option through credential profile configuration page in the Administration Console. The following table lists all the credential operations and the user checks that are performed depending on the type of operation.

Table 4-5   User Status Check

Operation / State	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
Reissue	Yes	Yes	No
Reset	Yes	Yes	No
Reset Validity	Yes	Yes	No
Delete Unsigned Attributes	No	No	No
Set Unsigned Attributes	No	No	No

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 credential:

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 username-password as well as ArcotID, while questions and corresponding answers are required for QnA credentials.

    NOTE: See "Credential Operations Summary" 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. (**Optional**) If you are implementing a callout, plug-in, or any other custom method for issuance operations, 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" for more information.

4. 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" and "Common Exceptions" 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 credential:

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 username-password as well as ArcotID, while questions and corresponding answers are required for QnA credentials.

    > **NOTE:** See "Credential Operations Summary" 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.   (**Optional**) If you are implementing a callout, plug-in, or any other custom method for issuance operations, 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" for more information.

4.   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" and "Common Exceptions" 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 username-password as well as ArcotID, while questions and corresponding answers are required for QnA credentials.

> **NOTE:** See "Credential Operations Summary" 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. (**Optional**) If you are implementing a callout, plug-in, or any other custom method for issuance operations, 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" for more information.

4. 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" and "Common Exceptions" 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

The Issuance APIs enables you to reset the credential. For example, you can reset the ArcotID 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 username-password as well as ArcotID, while questions and corresponding answers are required for QnA credentials.

    > **NOTE:** See "Credential Operations Summary" 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. (**Optional**) If you are implementing a callout, plug-in, or any other custom method for issuance operations, 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" for more information.

4. 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" and "Common Exceptions" 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 **get()** 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 username-password as well as ArcotID, while questions and corresponding answers are required for QnA credentials.

   > **NOTE:** See "Credential Operations Summary" 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. (**Optional**) If you are implementing a callout, plug-in, or any other custom method for issuance operations, 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" for more information.

4. 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" and "Common Exceptions" 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 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 username-password as well as ArcotID, while questions and corresponding answers are required for QnA credentials.

   > **NOTE:** See "Credential Operations Summary" 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. (**Optional**) If you are implementing a callout, plug-in, or any other custom method for issuance operations, 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" for more information.

4. 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" and "Common Exceptions" 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 username-password as well as ArcotID, while questions and corresponding answers are required for QnA credentials.

    NOTE: See "Credential Operations Summary" 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. (**Optional**) If you are implementing a callout, plug-in, or any other custom method for issuance operations, 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" for more information.

4. 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" and "Common Exceptions" 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 Notes

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

To reset the credential notes:

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 username-password as well as ArcotID, while questions and corresponding answers are required for QnA credentials.

   NOTE: See "Credential Operations Summary" 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. (**Optional**) If you are implementing a callout, plug-in, or any other custom method for issuance operations, 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" for more information.

4. Invoke the **resetNotes()** method of the **CredentialIssuance** interface to reset the credential notes.

   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" and "Common Exceptions" 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 Number of Questions

The number of questions that the user must set for QnA authentication might vary for every organization. To fetch the number of questions that must be set by the user:

1. (**Optional**) If you are implementing a callout, plug-in, or any other custom method for issuance operations, 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" 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.

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 username-password as well as ArcotID, while questions and corresponding answers are required for QnA credentials.

   > **NOTE:** See "Credential Operations Summary" 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. (**Optional**) If you are implementing a callout, plug-in, or any other custom method for issuance operations, 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" for more information.

4. 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" and "Common Exceptions" 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 of user, you need to implement the **setArcotIDUnsignedAttributes()** method:

NOTE: This operation is applicable *only* for ArcotID credential.

1. Use the **ArcotIDAttributes** class to set the ArcotID unsigned attributes.

2. (**Optional**) If you are implementing a callout, plug-in, or any other custom method for issuance operations, 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" for more information.

3. Invoke the **setArcotIDUnsignedAttributes()** method of the **CredentialIssuance** interface to set the ArcotID 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" and "Common Exceptions" 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 of a user, you need to implement the **deleteArcotIDUnsignedAttributes()** method:

> **NOTE:** This operation is only applicable for ArcotID credential.

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

1. (**Optional**) If you are implementing a callout, plug-in, or any other custom method for issuance operations, 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" for more information.

2. Invoke the **deleteArcotIDUnsignedAttributes()** method of the
    CredentialIssuance interface to delete the ArcotID 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" and "Common Exceptions" 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.

Table 4-6   Output Methods

Method	Description
Common Output Methods	
getCreateTime()	Fetches the time when the credential was created.
getLastFailedAuth AttemptTime()	Fetches the time when the last authentication attempt failed.
getLastSuccessAut hAttemptTime()	Fetches the time when the last authentication attempt succeeded.
getLastUpdatedTim e()	Fetches the time when the credential was updated last time.
getNotes()	Fetches the notes that are set for the credential.
getNumberOfFailed AuthAttempts()	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.

Table 4-6   Output Methods

Method	Description
getProfileVersion ()	Fetches the version number of the profile.
getStatus()	Fetches the status of the credential.
getUserName()	Fetches the name of the authenticating user.
getValidityEndTim e()	Fetches the date after which the credential expires.
getValidityStartT ime()	Fetches the date from when the credential is valid.
ArcotID Output Method	
getUnsignedAttrib utes()	Fetches the unsigned attributes of the ArcotID that the user has set.
QnA Output Method	
getQuestions()	Fetches the questions set for the user.
User-Name Password Output Method	
Output methods are not 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.
User Output Methods	
getCustomAttribut e	Fetches the value of the specified user attribute.
getCustomAttribut eMap()	Fetches the user attributes in a map, where the keys are the attribute names and the values are the corresponding attribute values.
getEmailId()	Fetches the email ID of the user.
getFirstName()	Fetches the first name of the user.
getLastName()	Fetches the last name of the user.
getMiddleName()	Fetches the middle name of the user.
getOrgName()	Fetches the name of the organization to which the user belongs.
getPAM()	Fetches the Personal Assurance Message (PAM) of the user.
getStatus()	Fetches the state of the user.
getTelephoneNumbe r()	Fetches the telephone number of the user.

Table 4-6   Output Methods

Method	Description
getUserName()	Fetches the name of the user.

Checking the Credential Status

To check the status of the credential, use the com.arcot.webfort.common.api package. The following table lists the different credential status.

Table 4-7   Credential Status

Status	Description
ACTIVE	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 WebFort Server successfully. **NOTE:** This status is applicable only for OTP.
DELETED	The credential of the user is deleted from the database.

State Transition

The following table lists the credential states and the transitions allowed between the states.

Table 4-8   Credential State Transition

To / From	Enabled	Locked	Disabled	Revoked	Verified
Enabled	Yes	Yes	Yes	Yes	Yes
Locked	Yes	Yes	Yes	Yes	No
Disabled	Yes	No	Yes	Yes	No
Revoked	No	No	No	Yes	No
Verified	Yes	No	Yes	Yes	No

Credential Operations and State

The following table lists all credential operations and whether each operation is allowed on a specific state of the credential or not. If the state of the credential changes after an operation, then the table also provides the next state of the credential.

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

Table 4-9   Credential Operation and States

State / Operation	Enabled	Locked	Disabled	Deleted	Verified (for OTP *only*)
Create	Not allowed	Not allowed	Not allowed	Enabled	Not allowed
Enable	Enabled	Enabled	Enabled	Not allowed	Not allowed
Disable	Disabled	Disabled	Disabled	Not allowed	Not allowed
Fetch	Allowed	Allowed	Allowed	Allowed	Allowed
Reset	Enabled	Enabled	Enabled	Not allowed	Enabled
Reset Validity	Allowed	Allowed	Allowed	Not allowed	Allowed
Reissue	Enabled	Enabled	Enabled	Not allowed	Enabled
Delete	Deleted	Deleted	Deleted	Deleted	Deleted
Delete Unsigned Attributes (for ArcotID *only*)	Allowed	Allowed	Allowed	Not allowed	Allowed
Set Unsigned Attributes (for ArcotID *only*)	Allowed	Allowed	Allowed	Not allowed	Allowed

Credential Operations Summary

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

- **ArcotID Operations**

- **Username-Password Operations**

- **Question and Answer Operations**

- **One-Time Password Operations**

ArcotID Operations

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

Table 4-10 ArcotID Operations

Operation	Input Required	Expected Output
Create	• User name (userName). • (Optional) Organization name (orgName). • ArcotID password (password). • (Optional) ArcotID attributes (signedAttributes and 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) Notes that you have maintained for each credential in your application (noteName, noteValue).	• CredentialOutput • TransactionDetails
Reset	• User name (userName). • (Optional) Organization name (orgName). • (Optional) Additional Input (AdditionalInput). • (Optional) Profile name of the credential.	• CredentialOutput • TransactionDetails
Fetch	• User name (userName). • (Optional) Organization name (orgName). • (Optional) Additional Input (AdditionalInput).	• CredentialOutput • TransactionDetails

Table 4-10 ArcotID Operations

Operation	Input Required	Expected Output
Reissue	• User name (`userName`). • (Optional) Organization name (`orgName`). • ArcotID password (`password`). • (Optional) ArcotID attributes (`signedAttributes` and `unsignedAttributes`). • (Optional) Additional Input (`AdditionalInput`). • (Optional) Profile name of the credential.	• `CredentialOutput` • `TransactionDetails`
Reset Validity	• User name (`userName`). • (Optional) Organization name (`orgName`). • (Optional) Additional Input (`AdditionalInput`). • Time when the validity of the credential ends (`validityEndTime`).	• `CredentialOutput` • `TransactionDetails`
Reset Notes	• User name (`userName`). • (Optional) Organization name (`orgName`). • (Optional) Additional Input (`AdditionalInput`). • Notes that you have maintained in your application (`noteName`, `noteValue`).	• `CredentialOutput` • `TransactionDetails`
Disable Enable Delete	• User name (`userName`). • (Optional) Organization name (`orgName`). • (Optional) Additional Input (`AdditionalInput`).	• `CredentialOutput` • `TransactionDetails`
Set unsigned attribute	• User name (`userName`). • (Optional) Organization name (`orgName`). • (Optional) Additional Input (`AdditionalInput`). • ArcotID attributes (`signedAttributes` and `unsignedAttributes`).	• `TransactionDetails`

Table 4-10   ArcotID Operations

Operation	Input Required	Expected Output
Delete unsigned attribute	• User name (userName). • (Optional) Organization name (orgName). • (Optional) Additional Input (AdditionalInput). • Array of ArcotID unsigned attributes.	• TransactionDet ails

Username-Password Operations

The following table provides the input and output information for username-password operations.

Table 4-11   Username-Password Operations

Operation	Input Required	Expected Output
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) Notes that you have maintained for each credential in your application (noteName, noteValue).	• CredentialResp onse
Reset	• User name (userName). • (Optional) Organization name (orgName). • (Optional) Additional Input (AdditionalInput). • (Optional) Profile name of the credential.	• CredentialResp onse
Fetch	• User name (userName). • (Optional) Organization name (orgName). • (Optional) Additional Input (AdditionalInput).	• CredentialResp onse

Table 4-11 Username-Password Operations

Operation	Input Required	Expected Output
Reissue	• User name (`userName`). • (Optional) Organization name (`orgName`). • Password (`password`). • (Optional) Additional Input (`AdditionalInput`).	• `CredentialResponse`
Reset Validity	• User name (`userName`). • (Optional) Organization name (`orgName`). • (Optional) Additional Input (`AdditionalInput`). • Time when the validity of the credential ends (`validityEndTime`).	• `CredentialResponse`
Reset Notes	• User name (`userName`). • (Optional) Organization name (`orgName`). • (Optional) Additional Input (`AdditionalInput`). • Notes that you have maintained in your application (`noteName`, `noteValue`).	• `CredentialResponse`
Disable Enable Delete	• User name (`userName`). • (Optional) Organization name (`orgName`). • (Optional) Additional Input (`AdditionalInput`).	• `CredentialResponse`

Question and Answer Operations

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

Table 4-12   QnA Operations

Operation	Input Required	Expected Output
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) Notes that you have maintained for each credential in your application (`noteName`, `noteValue`).	• `CredentialResponse`
Reset	• 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.	• `CredentialResponse`
Fetch	• User name (`userName`). • (Optional) Organization name (`orgName`). • (Optional) Additional Input (`AdditionalInput`).	• `CredentialResponse` • `questions`

Table 4-12 QnA Operations

Operation	Input Required	Expected Output
Reissue	• User name (`userName`). • (Optional) Organization name (`orgName`). • List of questions and answers (`question` and `answer`). • (Optional) Additional Input (`AdditionalInput`).	• `CredentialResponse`
Reset Validity	• User name (`userName`). • (Optional) Organization name (`orgName`). • (Optional) Additional Input (`AdditionalInput`). • Time when the validity of the credential ends (`validityEndTime`).	• `CredentialResponse`
Reset Notes	• User name (`userName`). • (Optional) Organization name (`orgName`). • (Optional) Additional Input (`AdditionalInput`). • Notes that you have maintained in your application (`noteName`, `noteValue`).	• `CredentialResponse`
Disable Enable Delete	• User name (`userName`). • (Optional) Organization name (`orgName`). • (Optional) Additional Input (`AdditionalInput`).	• `CredentialResponse`

One-Time Password Operations

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

Table 4-13   One-Time Password Operations

Operation	Input Required	Expected Output
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) Notes that you have maintained for each credential in your application. (noteName, noteValue).	• CredentialResp onse • Password
Fetch	• User name (userName). • (Optional) Organization name (orgName). • (Optional) Additional Input (AdditionalInput).	• CredentialResp onse
Reset	• User name (userName). • (Optional) Organization name (orgName). • (Optional) Additional Input (AdditionalInput).	• CredentialResp onse • Password
Reset Validity	• User name (userName). • (Optional) Organization name (orgName). • (Optional) Additional Input (AdditionalInput). • Time when the validity of the credential ends (validityEndTime).	• CredentialResp onse
Reset Notes	• User name (userName). • (Optional) Organization name (orgName). • (Optional) Additional Input (AdditionalInput). • Notes that you have maintained in your application (noteName, noteValue).	• CredentialResp onse

Table 4-13 One-Time Password Operations

Operation	Input Required	Expected Output
Disable	• User name (`userName`).	• `CredentialResponse`
Enable	• (Optional) Organization name (`orgName`).	
Delete	• (Optional) Additional Input (`AdditionalInput`).	

Chapter 5

Integrating ArcotID Client with Application

The **_ArcotID Client_** is a software that is used by the end user to sign the challenge provided by the WebFort Server. If you are planning to implement ArcotID-based authentication, then you must integrate ArcotID Client with application before you call ArcotID 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 Client. It covers the following topics:

- **ArcotID Client Overview**

- **Copying ArcotID Client Files**

- **ArcotID Client APIs**

ArcotID Client Overview

The ArcotID Client is used for signing the WebFort-issued challenge at the user end, but it also facilitates the download of the user's ArcotID. To support a wide variety of end user environments, the ArcotID 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 Client runs in any Web browser that has Adobe Flash Player (version 9 or higher) installed.

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

Signed Java Applet

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

> **NOTE:** When using the Arcot 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 Client Files

ArcotID 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**
- **For Java Signed Applet**

For Flash Client

The Flash client package contains the following files:

- **arcotclient.js**

  Contains the ArcotID Flash Client APIs.

- **ArcotIDClient.swf**

  Contains the ArcotID 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.

    > **NOTE:** Refer to *ArcotID Flash Client 6.0 API Guide* for more information on the Flash client configuration.

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 Client APIs

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

• **Downloading ArcotID**

• **Signing the Challenge**

Downloading ArcotID

To download the ArcotID 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 that has to be downloaded and the storage mode as the input parameters.

The ArcotID 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. An ArcotID can be stored in any of the following:

• **Hard Disk**

• **Universal Serial Bus (USB)**

• **Memory**

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

Signing the Challenge

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

> **NOTE:** Refer to *ArcotID Client 6.0 Reference Guide* for more information on the API details.

Chapter 6

Authenticating Users

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

- Initializing the Authentication SDK

- Preparing Additional Input

- ArcotID Authentication

- Questions and Answers Authentication

- Username-Password Authentication

- One-Time Password Authentication

- Authentication Tokens

- Verifying the Authentication Tokens

- Fetching the PAM

- Authentication Operations Summary

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.

Table 6-1   SDK Initialization by Using Map

Description	Input Values	Output Value

Initializes the Authentication SDK by using the provided map.	• map
	Key-value pair specifying the configuration information. The supported keys are:
	
	1. authentication.host.1
	   The IP address of the system where WebFort Server is available.
	
	2. authentication.port.1
	   The port on which the Authentication Native protocol is listening. Default value is 9742.
	
	• 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 parameters (transport, host, and port) in the
webfort.authentication.properties file are same as that of map. The
following table provides the details of the **init()** method.

Table 6-2   SDK Initialization by Using Properties File

Description	Input Values	Output Value

Initializes the Authentication SDK by using the properties file.	• location
	  The absolute path of the 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.

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.

Table 6-3   Releasing the API

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 WebFort's standard authentication capability by implementing callouts or plug-ins.

- You plan to write completely new custom authentication methods.

In all of these cases, you need to set the extra information that must be sent to WebFort Server in *name-value* pairs. WebFort'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 WebFort 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.

ArcotID Authentication

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

1. **ArcotID Download**

2. **ArcotID Authentication**

For successful ArcotID authentication, you must ensure that you have integrated ArcotID Client with application, as discussed in **Chapter 5, "Integrating ArcotID Client with Application," on page 61**.

> **NOTE:** The ArcotID download and authentication can be in multiple ways, see **Chapter 2, "Understanding WebFort WorkFlows"** for more information. This section focuses on the APIs that are used for these operations.

ArcotID Download

To perform ArcotID authentication, the ArcotID of the user has to be present on the system from where the authentication request is originating. If the ArcotID 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 is downloaded.

To download the ArcotID:

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

   See **"Preparing Additional Input"** for more information.

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

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

3. The user's ArcotID is set in the HTML or Java Server Page (JSP).

4. Invoke the **ImportArcotID()** client-side API to download the ArcotID from your application to the end user's system.

ArcotID Authentication

To perform ArcotID authentication:

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

    See "**Preparing Additional Input**" for more information.

2. Invoke the **getChallenge()** method of the **ArcotIDAuth** interface to retrieve
    the challenge form the WebFort 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 Client-side method, **SignChallengeEx()** to sign the
    challenge.

    The application collects the ArcotID password and the challenge is signed by the
    ArcotID Client using the ArcotID 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 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 WebFort Administration Console.

To perform the QnA authentication:

1. Invoke the **getQuestions()** method of the **QnAAuth** interface to retrieve the user's questions from the WebFort Server.

    This method returns an instance of the **QnAResponse** interface, which includes the questions to be asked, 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 WebFort Server.

    d. answerQuestion

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

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

    See **"Preparing Additional Input"** for more information.

4. Invoke the **verifyAnswers()** method of the **QnAAuth** interface by passing the
    AuthQnAInfo object created in **Step 2 on page 73** 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.

Username-Password Authentication

The authentication API provides the **PasswordAuth** interface to perform the traditional *username-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.

To perform username-password authentication:

1. Implement the logic to collect the user's password.

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

   See "Preparing Additional Input" 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.

   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 WebFort Server. WebFort 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 callout or plug-in, then invoke the **setAdditionalInput()** method in the **AdditionalInput** class to obtain an object that implements the class.

    See "**Preparing Additional Input**" 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.

Authentication Tokens

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

By using the Authentication API, 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 **AuthTokenType** class specifies the return token type and supports the following types of tokens:

- **Native Tokens**

  Specify this type when Arcot-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 WebFort can only be interpreted by the WebFort Server, but the SAML tokens issued by the WebFort Server can be interpreted by any other authentication system. WebFort supports **1.1** and **2.0** versions of SAML:

  - **SAML 1.1 Tokens**

    Specify this type of token when you are using custom (non-WebFort) 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-WebFort) 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.

Verifying the Authentication Tokens

WebFort 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 these token for Single Sign-On, wherein you authenticate the user once and allow them to use multiple resources using the authentication token.

To verify if a token is valid or not:

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

   See "**Preparing Additional Input**" for more information.

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

   This methods 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.

To obtain the PAM of a user:

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

    See "**Preparing Additional Input**" for more information.

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

    This methods 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.

Table 6-4   Authentication Operations

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 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 default organization. • Question and Answers for authentication (`qnaInfo`) • (Optional) Additional Input (`AdditionalInput`) • (Optional) Authentication token type (`AuthTokenType`)	

Table 6-4   Authentication Operations

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

Chapter 7

Using Custom APIs

WebFort provides Custom APIs to enable you to add new credentials. You can use these APIs to create and manage these credentials and also use them for authentication. The Custom APIs can either be used to:

*   Add completely new authentication methods, for example, certificate-based or hardware-based.

*   Support WebFort authentication methods along with the method that is derived based on the WebFort authentication methods. For example, you can use WebFort OTP in parallel with the authentication method that is based on WebFort OTP.

> **NOTE:** For WebFort to support the Custom APIs and perform the operations mentioned in this chapter, you must configure the plug-in or callout that provides these functions. Refer *Arcot WebFort 6.0 Business Logic Extension Guide* to for more information on this.

This chapter discusses both Issuance and Authentication APIs for custom credentials. It covers the following topics:

*   **Issuance Operations**

*   **Authentication Operations**

Issuance Operations

The following issuance operations are supported for the custom credential:

- **Creating Credential**
- **Enabling Credential**
- **Disabling Credential**
- **Resetting Credential**
- **Reissuing Credential**
- **Resetting Credential Validity**
- **Fetching Credential Details**
- **Deleting Credentials**

Creating Credential

The **com.arcot.webfort.issuance.api** package provides the **CustomIssuance** interface that contains the method to create the credential

Perform the following steps to create credential:

1. Use the **CustomInput** class that encapsulates credential information.

2. Invoke the **setAdditionalInput()** method in the **AdditionalInput** class to obtain an object that implements the class.

   This class provides the additional information that is set as a name-value pair. See **"Preparing Additional Input"** for more information.

3. Invoke the **create()** method of the **CustomIssuance** interface to create the credentials.

   This method returns an instance of the **CustomResponse** interface, which specifies the credential and transaction details.

Disabling Credential

Perform the following steps to disable credential:

1.   Use the **CustomInput** class that encapsulates credential information.

2.   Invoke the **setAdditionalInput()** method in the **AdditionalInput** class to obtain an object that implements the class.

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

3.   Invoke the **disable()** method of the **CustomIssuance** interface to disable the credentials.

     This method returns an instance of the **CustomResponse** interface, which specifies the credential and transaction details.

Enabling Credential

Perform the following steps to enable credential:

1.   Use the **CustomInput** class that encapsulates credential information.

2.   Invoke the **setAdditionalInput()** method in the **AdditionalInput** class to obtain an object that implements the class.

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

3.   Invoke the **enable()** method of the **CustomIssuance** interface to enable the credentials.

     This method returns an instance of the **CustomResponse** interface, which specifies the credential and transaction details.

Resetting Credential

Perform the following steps to reset the credential:

1.   Use the **CustomInput** class that encapsulates credential information.

2.   Invoke the **setAdditionalInput()** method in the **AdditionalInput** class to obtain an object that implements the class.

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

3.   Invoke the **resetCredential()** method of the **CustomIssuance** interface to reset the custom credential.

This method returns an instance of the **CustomResponse** interface, which specifies the credential and transaction details.

Reissuing Credential

Perform the following steps to reissue the credential:

1. Use the **CustomInput** class that encapsulates credential information.

2. Invoke the **setAdditionalInput()** method in the **AdditionalInput** class to obtain an object that implements the class.

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

3. Invoke the **reissue()** method of the **CustomIssuance** interface to reissue the custom credential.

    This method returns an instance of the **CustomResponse** interface, which specifies the credential and transaction details.

Resetting Credential Validity

Perform the following steps to reset the validity of the credential:

1. Use the **CustomInput** class that encapsulates credential information.

2. Invoke the **setAdditionalInput()** method in the **AdditionalInput** class to obtain an object that implements the class.

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

3. Invoke the **resetValidity()** method of the **CustomIssuance** interface to reset the validity of the custom credential.

    This method returns an instance of the **CustomResponse** interface, which specifies the credential and transaction details.

Fetching Credential Details

Perform the following steps to fetch the credential details:

1. Use the **CustomInput** class that encapsulates credential information.

2. Invoke the **setAdditionalInput()** method in the **AdditionalInput** class to obtain an object that implements the class.

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

3. Invoke the **fetch()** method of the **CustomIssuance** interface to fetch the custom credential details.

   This method returns an instance of the **CustomResponse** interface, which specifies the credential and transaction details.

Deleting Credentials

Perform the following steps to delete the credential:

1. Use the **CustomInput** class that encapsulates credential information.

2. Invoke the **setAdditionalInput()** method in the **AdditionalInput** class to obtain an object that implements the class.

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

3. Invoke the **delete()** method of the **CustomIssuance** interface to delete the custom credential.

   This method returns an instance of the **CustomResponse** interface, which specifies the credential and transaction details.

Authentication Operations

The **com.arcot.webfort.authentication.api** package provides the **CustomAuth** interface that contains the methods to authenticate password-based and challenge-response-based authentication methods.

Password-Based Authentication

To perform the password-based authentication:

1. You must collect the user password.

2. Invoke the **setAdditionalInput()** method in the **AdditionalInput** class to obtain an object that implements the class.

   This class provides the additional information that is set as a name-value pair. See **"Preparing Additional Input"** for more information.

3. Invoke the **verifyCredential()** method of the **CustomAuth** 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.

   This method returns an instance of the **CustomAuthResponse** interface, which provides the transaction details, user details, and token information.

Challenge-Response-Based Authentication

To perform the challenge-response-based authentication:

1. Invoke the **setAdditionalInput()** method in the **AdditionalInput** class to obtain an object that implements the class.

   This class provides the additional information that is set as a name-value pair. See **"Preparing Additional Input"** for more information.

2. Invoke the **getChallenge()** method of the **CustomAuth** interface to receive the challenge.

   This method returns an instance of the **CustomChallengeResponse** interface, which specifies the transaction details and also gets the challenge from the server.

3. Perform the authentication depending on the type of credential used. For example, if it is challenge-response authentication, then the user will sign a challenge, or if it is a question and answers authentication, then the user will provide answers for authentication.

4. Invoke the **verifyCredential()** method of the **CustomAuth** interface to verify the credential. 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 **CustomAuthResponse** interface, which provides the transaction details, user details, and token information.

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, WebFort Server validates the data that it receives from the APIs. Table A-1 explains the criteria that the WebFort Server uses to validate this input data.

> **NOTE:** Attribute length mentioned in the following table corresponds to the character length.

> **NOTE:** Attribute ID is referred to as `paramName` in the Java APIs.

Table A-1   Attribute Validation Checks

Attribute	Attribute ID	Validation Criteria
User name	USER_NAME	User name is non-empty.
		User name length is between 1 and 256 characters.
User status	USER_STATUS	Check whether the user status is active or not: • Active state -> USER_STATUS_ACTIVE • Disabled state -> USER_STATUS_DISABLED

Table A-1   Attribute Validation Checks

Attribute	Attribute ID	Validation Criteria
Email ID	EMAIL_ID	Email ID is non-empty.
		Email ID length is between 1 and 128 characters.
		Does not contain whitespace characters except SPACE (ASCII 0-31).
		Check the following: • @ should come before period (.) • It should contain one @ character • It should contain at least one period (.)
First name	FIRST_NAME	First name is non-empty.
		First name length is between 1 and 32 characters.
		Does not contain whitespace characters except SPACE (ASCII 0-31).
Last name	LAST_NAME	Last name is non-empty.
		Last name length is between 1 and 32 characters.
		Does not contain whitespace characters except SPACE (ASCII 0-31).
Personal Assurance Message	PAM	PAM is non-empty.
		PAM length is between 1 and 128 characters.
		Does not contain whitespace characters except SPACE (ASCII 0-31).
Start time	START_TIME	Check for the valid date format.
End time	END_TIME	Check for the valid date format.
Organizatio n name	ORG_NAME	Organization name is non-empty.
		Organization name length is between 1 and 64 characters.
		Does not contain whitespace characters except SPACE (ASCII 0-31).
Locale name	LOCALE_NAME	Locale name is non-empty.
		Check whether the locale name is according to ISO set of locales.

Table A-1   Attribute Validation Checks

Attribute	Attribute ID	Validation Criteria
Authentication user password	AUTH_USER_PASSWORD	User password is non-empty.
		User password length is between 1 and 64 characters.
		Check user password against a set of strings.
		Does not contain whitespace characters except SPACE (ASCII 0-31).
Password maximum length	PWD_MAX_LENGTH	Minimum value of password maximum length > 4 characters.
		Maximum value of password maximum length < 64 characters.
Password minimum length	PWD_MIN_LENGTH	Minimum value of password minimum length > 4 characters.
		Maximum value of password minimum length < 64 characters.
Question	AUTH_QUESTIONS	Question is non-empty.
		Question length is between 1 and 64 characters.
		Does not contain whitespace characters except SPACE (ASCII 0-31).
Answer	AUTH_ANSWERS	Answer is non-empty.
		Answer length is between 1 and 64 characters.
		Does not contain whitespace characters except SPACE (ASCII 0-31).
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

Table A-1   Attribute Validation Checks

Attribute	Attribute ID	Validation Criteria
Password	Password	Password is non-empty.
		Password length is between 1 and 64 characters.
		Does not contain whitespace characters except SPACE (ASCII 0-31).
OTP maximum length	OTP_MAX_LENGTH	OTP length is between 4 and 64 characters.

Appendix B

WebFort Logging

WebFort server logs contain transaction data and flow details for various types of operations. These logs can be used to debug various types of issues, such as incorrect configurations, incorrect transaction data, and other failures. This appendix describes how to understand WebFort Server startup and transaction logs.

For few operations you might not receive an error message, for example, during server startup. Even if you receive an error message, you may have to use the log file as one of your tools to determine the issue.

The following topics are discussed in this appendix:

- **About the Log Files**

- **Format of the WebFort Log Files**

- **Supported Severity Levels**

About the Log Files

WebFort allows you to change the logging parameters as well as logging location and directory information for the log files. The typical logging configuration options that you can change in these files include:

- **Specifying log file name and path:** WebFort enables you to specify the directory for writing the log files and storing the backup log files. In addition to these, you can also control whether you want to enable trace logging.

- **Log file size:** The maximum number of bytes the log file can contain. When the log files reach this size, a new file with the specified name is created and the old file is moved to the backup directory.

- **Using log file archiving:** WebFort 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:** WebFort 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" for more information on the supported log levels.

- **Specifying time zone information:** WebFort enables you to either use the local time zone for time stamping the logged information or use GMT for the same.

The WebFort log files can be categorized as startup log and transaction logs. The default location of these file is:

Windows: `<install_location>`\Arcot Systems**logs**\

UNIX-Based: `<install_location>`/arcot/**logs**/

The following table explains the WebFort log files.

Table B-1   Log Files

Log File Type	Log File Name	Description
Startup Log	**arcotwebfortstartup.log** (**Startup Log**)	When you start the WebFort Server, it records all start-up (or boot) actions in this file. The information in this file is very useful in identifying the source of the problems if the WebFort service does not start up.

Table B-1   Log Files

Log File Type	Log File Name	Description
Transaction Logs	**arcotwebfort.log** (Server Log)	WebFort records all requests processed by the server in the **arcotwebfort.log** file. 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.
	arcotwebfortstas.log (Statistics Log)	WebFort uses this file for logging statistics.
	arcotwebfort.log (Trace Log)	WebFort 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:.

Format of the WebFort Log Files

Table B-2 describes the format of the entries in the WebFort loggers:

Table B-2 WebFort Logging Format

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" on page 99** for more information.
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.

Supported Severity Levels

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

Table B-3 describes the log levels in the *decreasing* order of severity.

Table B-3   WebFort Log Levels (in 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 WebFort service.
1	**WARNING**	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:
		• Server state, such as start, stop, and restart.
		• Server properties.
		• State of services.
		• State of a processes on the Server.
		• Requested transaction.
		• Operation involved.
		• Result of the transaction.
3	**DEBUG**	Use this log level for logging detailed information for debugging purposes. This might include changes in Server states.

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 INFO, then messages with log levels of FATAL, WARNING level are also captured.

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

FATAL

```
08/13/09 14:32:51.889 FATAL STARTUP     00003820 00WFMAIN - Arcot WebFort
Authentication Service SHUTDOWN due to initialization failure!
```

```
08/13/09 14:33:58.154 FATAL STARTUP     00003612 00WFMAIN - [7]: Database
password could not be obtained from securestore.enc for [wf-60-st1-mssql-test]

08/13/09 14:33:58.154 FATAL STARTUP     00003612 00WFMAIN - Arcot WebFort
Authentication Service SHUTDOWN due to initialization failure!
```

WARNING

```
08/13/09 14:13:31.451 WARN  SVRMGMT_WS   00005808 00055503 -
ArDBConnection::GetDBDiagnosis: SQL State:01000, Native Code: 2746, ODBC code:
[Arcot Systems][ODBC SQL Server Driver][DBNETLIB]ConnectionWrite (send()).

08/13/09 14:13:31.451 WARN  SVRMGMT_WS   00005808 00055503 -
ArDBConnection::GetDBDiagnosis: SQL State:01000, Native Code: 2746, ODBC code:
[Arcot Systems][ODBC SQL Server Driver][DBNETLIB]ConnectionWrite (send()).SQL
State:08S01, Native Code: B, ODBC code: [Arcot Systems][ODBC SQL Server
Driver][DBNETLIB]General network error. Check your network documentation.

08/13/09 14:13:31.451 WARN  SVRMGMT_WS   00005808 00055503 -
ArDBPoolManagerImpl::isKnownFailure: Error state [08S01] is detected as known
failure(type:0)!

08/13/09 14:13:31.451 WARN  SVRMGMT_WS   00005808 00055503 -
ArDBPoolManagerImpl::reportQueryFailure: Query failure is detected as DBFO for
primary DSN [wf-60-st1-mssql] and context [70]. Marking it bad.

08/13/09 14:13:31.451 WARN  MONITOR      00004984 0MONITOR -
ArDBPoolManagerImpl::recoverPool: [primary] pool [wf-60-st1-mssql] is marked bad.
Will try to clean and connect...

08/13/09 14:13:31.467 WARN  SVRMGMT_WS   00005808 00055503 - ArDBM::Caught
ArcotException in _DbOp!. err : [Arcot Exception,Error: All database pools are
inactive]
```

INFO

```
08/13/09 11:52:17.493 INFO  STARTUP      00003504 00WFMAIN -
-----------------------------------------------------------------
08/13/09 11:52:17.493 INFO  STARTUP      00003504 00WFMAIN - Listing : [Server
startup]
08/13/09 11:52:17.493 INFO  STARTUP      00003504 00WFMAIN -
-----------------------------------------------------------------
08/13/09 11:52:17.493 INFO  STARTUP      00003504 00WFMAIN - Timezone
information........................: [08/13/09 11:52:17 India Standard Time]
08/13/09 11:52:17.493 INFO  STARTUP      00003504 00WFMAIN -
ARCOT_HOME.................................: [D:\ArcotInstalls2\Arcot Systems]
08/13/09 11:52:17.493 INFO  STARTUP      00003504 00WFMAIN - Server
ProcessID..............................: [4316]
08/13/09 11:52:17.493 INFO  STARTUP      00003504 00WFMAIN -
-----------------------------------------------------------------
```

DEBUG

```
08/12/09 21:31:48.236 DEBUG SVRMGMT_WS   00004052 00050516 -
ArDBPoolManagerImpl::getLockedDBConnection: Returning DBPool [01E11FF0]

08/12/09 21:31:48.236 DEBUG SVRMGMT_WS   00004052 00050516 - TRACE: Releasing
Cache read lock on [01E2C8D8]

08/12/09 21:31:48.236 DEBUG SVRMGMT_WS   00004052 00050516 - TRACE: Released Cache
read lock on [01E2C8D8]

08/12/09 21:31:48.236 DEBUG SVRMGMT_WS   00004052 00050516 - TRACE:
CallTrace::Leaving : [ArDBPoolManagerImpl::selectAnActivePool]. time : 0
```

```
08/12/09 21:31:48.236 DEBUG SVRMGMT_WS   00004052 00050516 - TRACE:
CallTrace::Entering : [ArDBPool::getLockedDBConnectionConst]

08/12/09 21:31:48.236 DEBUG SVRMGMT_WS   00004052 00050516 - TRACE:
ArDBPool::getLockedDBConnection [(primary)] : GotContext [86], [127] more
connections available
```

Appendix C

Additional Settings

This appendix discusses the following topics:

- **Configuring Multiple WebFort Server Instances**
- **Setting up SSL**

Configuring Multiple WebFort Server Instances

To configure Java SDKs with multiple WebFort Server instances, you must edit the properties file. By default, the file provides entries to configure 1 WebFort Server instance. These entries are appended with 1, indicating that only 1 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 WebFort 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\java\properties`

Unix-Based Platforms: `<install_location>/arcot/sdk/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"** 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 WebFort Server.

4. Set the value of `port.<n>` parameter to the port number on which the Authentication Native or the Transaction Web Services protocol is listening

Setting up SSL

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

Figure C-1 WebFort Communication Modes

Before you enable SSL communication between Java SDKs and WebFort Server, you must,

1. Obtain a digital certificate from a trusted Certificate Authority.

2. Expose your application over an HTTPS-enabled server port.

To set up SSL between Java SDK and WebFort Server:

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

Windows: `<install_location>\Arcot Systems\sdk\java\properties`
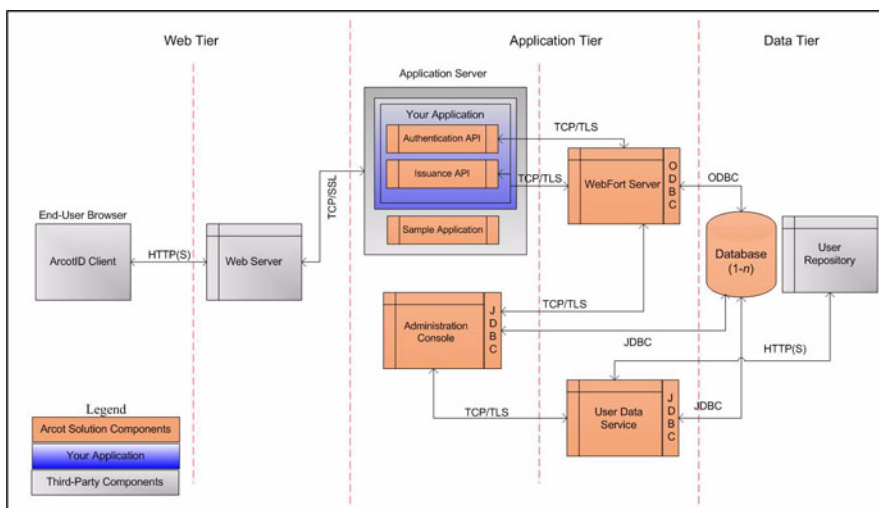
Unix-Based Platforms: `<install_location>/arcot/sdk/java/properties`

2. Set the value of `<SDK name>.transport.<n>` parameter to `ssl`.

3. Set the value of `<SDK name>.serverCACertPEMPath` parameter to the path of server CA certificate file. The file must be in `.PEM` format.

4. Set the value of `<SDK name>.clientCertKeyP12Path` parameter to the path of client certificate file. The file must be in `.p12` format.

5. Set the value of `<SDK name>.clientCertKeyPassword` parameter to the password of p12 store.

6. Save and close the file.

Appendix D

SDK Exceptions

This chapter lists all exceptions that are returned by the WebFort 6.0 SDKs. These exceptions have been categorized as:

- **Common Exceptions**
- **Issuance Exceptions**
- **Authentication Exceptions**

Common Exceptions

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

Table D-1   Common Exception Classes of
com.arcot.webfort.common.api.exception

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.
`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 WebFort 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.

Table D-1   Common Exception Classes of
com.arcot.webfort.common.api.exception

Classes	Exception Returned By	Description
UserNotFoundException	Server	This exception is returned if the user trying to perform the operation is not enrolled in WebFort.

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 WebFort Server.

Table D-2   Issuance Exception Classes of
                com.arcot.webfort.issuance.api.exception

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.
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 WebFort Server.

Table D-3   Authentication Exception Classes of
com.arcot.webfort.authentication.api.exception

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.

Appendix E

WebFort Error Codes

This appendix categorizes WebFort error codes as:

- **SDK Codes**
- **Server Codes**

SDK Codes

Table E-1 lists the SDK response codes, cause for failure, and solution wherever applicable.

Table E-1   SDK Response Codes and Cause for Failure

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 `SDKNotInitializedException`)	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 `SDKAlreadyInitializedException`)	SDK is already initialized.	**Possible Cause:** User is trying to initialize the SDK that has already been initialized.

Table E-1   SDK Response Codes and Cause for Failure

SDK Response Code	Description	Possible Cause for Failure
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.
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 TLS 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.

Table E-1   SDK Response Codes and Cause for Failure

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.

Table E-2   Response and Reason Codes

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.
	2000	Database is not running.	**Possible Cause:** Database is not running. **Solution:** Start the database. **Possible Cause:** Connection between the server and database is not complete. **Solution:** Establish the connection between server and database again.

Table E-2   Response and Reason Codes

Response Code	Reason Code	Description	Possible Cause for Failure
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.
1050	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 **"Input Data Validations"** for the supported parameter values.

Table E-2   Response and Reason Codes

Response Code	Reason Code	Description	Possible Cause for Failure
1050	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 "Input Data Validations" for the supported parameter values.
	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 "Input Data Validations" for the supported parameter values.
	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 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 "Input Data Validations" for the supported parameter values.

Table E-2   Response and Reason Codes

Response Code	Reason Code	Description	Possible Cause for Failure
1050	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 **"Input Data Validations"** for the supported parameter values.
	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 **"Input Data Validations"** 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.

Table E-2   Response and Reason Codes

Response Code	Reason Code	Description	Possible Cause for Failure
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.
	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.

Table E-2   Response and Reason Codes

Response Code	Reason Code	Description	Possible Cause for Failure
1050	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.
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.
1100	0	Organization is not found.	**Possible Cause:** 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 the server cache. 3. Check if the name of the organization passed is correct.

Table E-2   Response and Reason Codes

Response Code	Reason Code	Description	Possible Cause for Failure
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.
1150	0	User status is not active.	**Possible Cause:** User status is not active. **Solution:** Activate the user.
1151	0	User already exists.	**Possible Cause:** User already present in the system. **Solution:** Create the user with different user name or provide the user details correctly.

Table E-2   Response and Reason Codes

Response Code	Reason Code	Description	Possible Cause for Failure
1152	0	Credential is invalid.	**Possible Cause:** Credential already present for the user. **Solution:** Do not create a credential that already exists for the user.
5500	0	Processor is invalid. Note that processor refers to authentication mechanism.	**Possible Cause:** The mechanism requested is not supported by the system. **Solution:** Use mechanisms supported by the systems.
5604	0	Organization is invalid.	**Possible Cause:** The provided organization name is not valid. **Solution:** You must provide a valid organization name.
5605	0	SSL trust store organization name is invalid.	**Possible Cause:** The provided organization name is not valid. **Solution:** You must provide a valid organization name.
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.

Table E-2   Response and Reason Codes

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.
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.
5706	0	Credential is reissued.	**Possible Cause:** Credential is reissued.
5707	0	Provided credential details are incorrect.	**Possible Cause:** The credential details provided by the user are incorrect. **Solution:** Provide the credential details correctly.
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.
5801	0	Credential already present for the user.	**Possible Cause:** Credential already exists for the user.

Appendix F

WebFort Sample Application

WebFort 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 WebFort Server. See section, **"Configuring Sample Application,"** for information on how to do this.

The following sections then walk you through the sample tasks that you can perform by using the Sample Application:

- **Selecting ArcotID Client**

- **Demonstrating User Operations**

- **Demonstrating ArcotID Operations**

- **Demonstrating QnA Operations**

- **Demonstrating OTP Operations**

- **Demonstrating User-Password Operations**

Configuring Sample Application

The **Setup** page enables you to set the WebFort 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>:<port>/webfort-6.0-sample-application`

    The WebFort 6.0 Sample Application page appears.

2. From the sidebar, click the **Setup** link to display the WebFort Server Connectivity page.

3. Specify the values for the configuration parameters listed in **Table F-1**.

Table F-1   Configuration Parameters

Field	Default Value	Description
Logger Configuration		
Log File Path	`./arcotwebfortsample app.log`	The absolute path to the Sample Application log file.
Server Configuration		
IP Address	localhost	The host name or the IP address where the WebFort Server is available.
Port	**Authentication Service:** `9742` **Issuance Service:** `9744`	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 WebFort Server.

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

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

1. Click the **[+]** sign preceding **Additional Server Configuration**.

2. Specify the **IP Address** and **Port** connection parameters.

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

Selecting ArcotID Client

Before you perform any ArcotID-related operations, as discussed in "Demonstrating ArcotID Operations" on page 132, you must choose the appropriate type of ArcotID Client that you want to use, along with the required storage medium where you want to store the downloaded ArcotID.

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

To select the ArcotID Client:

1. From the sidebar, click the **Select Client** link.

   The Select ArcotID Client page appears.

2. Select the type of ArcotID Client that you want to use from **Choose ArcotID Client** section.

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

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

   See section, "Downloading ArcotID" on page 65 for more information on the supported download types.

4. Click **Select** to store the settings.

Demonstrating User Operations

The Sample Application demonstrates how the APIs are used for user operations. You can do the following by using the Sample Application:

- **Creating Users**
- **Updating User Information**
- **Fetching User Information**

Creating Users

To create a user:

1. Under the **User** sidebar menu, click the **Create** link.

   The Create User page appears.

2. Specify the user information in the first section to create the user.

3. Specify the **Custom Attributes** that you want to maintain for the user.

   This section enables you to specify additional attributes for the UserInput class.

4. Set the **Additional Inputs,** such as locale information, that you want to maintain for the user.

5. Click **Create**.

Updating User Information

To update the user information:

1. Under the **User** sidebar menu, click the **Update** link.

   The Update User page appears.

2. Enter the user information in this page.

3. Click **Update** to update the user.

Fetching User Information

To fetch the details for the user:

1. Under the **User** sidebar menu, click the **Fetch** link.

   The Fetch User page appears.

2. Specify the information based on which you want to fetch the user details.

3. Click **Fetch**.

Demonstrating ArcotID Operations

Sample Application demonstrates how the WebFort SDKs are used to create, download, and authenticate ArcotID. You can do the following user operations using the Sample Application:

- **Creating ArcotID**

- **Downloading the ArcotID**

- **Authenticating Using the ArcotID**

- **Fetching the ArcotID Details**

Creating ArcotID

To create the ArcotID for a user:

1. Under the **ArcotID** sidebar menu, click the **Create** link.

    The Create ArcotID page appears.

2. Enter the user details, such as **User Name**, **Organization Name**, **ArcotID Password**, and the **Profile Name**.

    > **NOTE:** Profile name is optional, if you use this, then ensure it is already created and assigned to the organization.

3. Specify the **Unsigned Attributes** that you want to set for the ArcotID.

    These attributes enable you to specify additional information for the ArcotID.

4. Enter the **Notes** that you want to maintain for the ArcotID.

    For example, you can use this section to note that the user cannot download ArcotID on more than two systems.

5. Set the **Additional Inputs**, such as locale information for the current transaction.

6. Click **Create**.

Downloading the ArcotID

To download the ArcotID from the WebFort Server:

1. Under the **ArcotID** sidebar menu, click the **Download** link.

    The Download ArcotID Client page appears.

2. Specify the **User Name** and the **Organization** to which the user belongs.

3. Set the **Additional Inputs**, such as locale information for the current transaction.

4. Click **Download** to download the ArcotID.

Authenticating Using the ArcotID

To authenticate to WebFort using the ArcotID:

1. Under the **ArcotID** sidebar menu, click the **Authenticate** link.

    The ArcotID Authentication page appears.

2. Specify relevant information in the **User Name**, **Organization**, and the **ArcotID Password** fields.

3. Optionally, select the **Token Type** that has to be returned after successful authentication.

4. Optionally, enter the authentication and the SAML policy names, in the **Authentication Policy Name** and **SAML Policy Name** fields, respectively.

    > **NOTE:** If you are using these fields, then ensure the policies are already created and assigned to the organization.

5. Set the **Additional Inputs**, such as locale information for the current transaction.

6. Click **Authenticate** to authenticate to the configured WebFort Server.

    After successful authentication, the Sample Application returns credential details, transaction ID, notes, and authentication token.

Fetching the ArcotID Details

To fetch the ArcotID details for the specified user:

1. Under the **ArcotID** sidebar menu, click the **Fetch** link.

    The Fetch ArcotID page appears.

2. Specify relevant information in the **User Name**, **Organization**, and the **ArcotID Password** fields.

3. Enter the ArcotID **Additional Inputs**, if required.

4. Click **Fetch** to retrieve the ArcotID details.

    The Sample Application returns the unsigned attributes and notes for the credential.

Demonstrating QnA Operations

Sample Application demonstrates how the WebFort SDKs are used to create, authenticate, and fetch QnA for users. You can do the following user operations using the Sample Application:

*   **Creating QnA**

*   **Authenticating Using QnA**

*   **Fetching the QnA Details**

Creating QnA

To create the QnA for a user:

1.   Under the **QnA** sidebar menu, click the **Create** link.

     The Create Questions & Answers (QnA) page appears.

2.   Optionally, specify the **Organization Name** and the **Profile Name** for the user.

         NOTE: Profile name is optional, if you use this, then ensure it is already created and assigned to the organization.

3.   Enter the **User Name** for whom you want to create the QnA.

4.   Click the **Fetch** button against the **Number Of Questions Configured For Organization** field.

     The Specify Question & Answers section appears. The rows displayed in this section depends on the QnA profile configured for the organization.

5.   Depending on the QnA configuration, specify minimum unique questions and their corresponding answers in the **Question** and **Answer** fields, respectively.

6.   Enter the **Notes** that you want to maintain for the QnA.

7.   Set the **Additional Inputs**, such as locale information for the current transaction.

8.   Click **Create**.

Authenticating Using QnA

To authenticate to WebFort using QnA:

1. Under the **QnA** sidebar menu, click the **Authenticate** link.

    The QnA Authentication page appears.

2. Enter the **User Name** who you want to authenticate using QnA.

3. Optionally, specify the **Organization** for the user.

4. Click the **Fetch Questions** button to retrieve the user's questions.

    The Specify Question & Answers section appears.

5. Specify the answers for the corresponding questions in the **Answer** fields.

6. Specify the **Additional Inputs**, such as locale information for the current transaction.

7. Click **Authenticate** to authenticate to the configured WebFort Server.

    After successful authentication, the Sample Application returns credential details, transaction ID, notes, and authentication token.

Fetching the QnA Details

To fetch the QnA details for the specified user:

1. Under the **QnA** sidebar menu, click the **Fetch** link.

    The Fetch QnA page appears.

2. Enter the **User Name** whom you want to authenticate using QnA.

3. Optionally, specify the **Organization** for the user.

4. Specify the **Additional Inputs**, such as locale information for the current transaction.

5. Click **Fetch** to retrieve the QnA details.

Demonstrating OTP Operations

Sample Application demonstrates how the WebFort SDKs are used to create, authenticate, and fetch OTP for users. You can do the following user operations using the Sample Application:

- **Creating OTP**
- **Authenticating Using the OTP**
- **Fetching the OTP Details**

Creating OTP

To create the OTP for a user:

1. Under the **OTP** sidebar menu, click the **Create** link.

    The Create OTP page appears.

2. Enter the **User Name** for whom you want to create the OTP.

3. Optionally, specify the **Organization** and the **Profile Name** for the user.

    > **NOTE:** Profile name is optional, if you use this, then ensure it is already created and assigned to the organization.

4. Enter any **Notes** that you might want to maintain for the OTP.

5. Set the **Additional Inputs**, such as locale information for the current transaction.

6. Click **Create**.

    The Sample Application returns the OTP for the user.

Authenticating Using the OTP

To authenticate to WebFort using OTP:

1. Under the **OTP** sidebar menu, click the **Authenticate** link.

    The OTP Authentication page appears.

2. Enter the **User Name** who you want to authenticate using OTP.

3. Optionally, specify the **Organization** for the user.

4. Enter the **OTP** issued to the specified user, see **"Creating OTP"**.

5. Optionally, select the **Token Type** that has to be returned after successful authentication and enter the authentication and the SAML policy names, in the **Authentication Policy Name** and **SAML Policy Name** fields, respectively.

> **NOTE:** If you are using these fields, then ensure the policies are already created and assigned to the organization.

6. Specify the **Additional Inputs**, such as locale information for the current transaction.

7. Click **Authenticate** to authenticate to the configured WebFort Server.

   After successful authentication, the Sample Application returns credential details, transaction ID, notes, and authentication token.

Fetching the OTP Details

To fetch the OTP details for the specified user:

1. Under the **OTP** sidebar menu, click the **Authenticate** link.

   The Fetch OTP page appears.

2. Enter the **User Name** whom you want to authenticate using QnA.

3. Optionally, specify the **Organization** for the user.

4. Specify the **Additional Inputs**, such as locale information for the current transaction.

5. Click **Fetch** to retrieve the OTP details.

Demonstrating User-Password Operations

Sample Application demonstrates how the Authentication SDK is used to create, authenticate, and fetch User-Password for users. You can do the following user operations using the Sample Application:

- **Creating User-Password**

- **Authenticating Using User-Password**

- **Fetching User-Password Details**

Creating User-Password

To create the Password for a user:

1. Under the **User Password** sidebar menu, click the **Create** link.

    The Create User-Password page appears.

2. Enter the **User Name** and **Password** for whom you want to create the User-Password.

3. Optionally, specify the **Organization** and the **Profile Name** for the user.

    > **NOTE:** Profile name is optional, if you use this, then ensure it is already created and assigned to the organization.

4. Enter any **Notes** that you might want to maintain for the User-Password.

5. Set the **Additional Inputs**, such as locale information for the current transaction.

6. Click **Create**.

Authenticating Using User-Password

To authenticate to WebFort using User-Password mechanism:

1. Under the **User Password** sidebar menu, click the **Authenticate** link.

    The User-Password Authentication page appears.

2. Enter the **User Name** who you want to authenticate using OTP.

3. Optionally, specify the **Organization** for the user.

4. Enter the **Password** issued to the specified user.

5. Optionally, select the **Token Type** that has to be returned after successful authentication and enter the authentication and the SAML policy names, in the **Authentication Policy Name** and **SAML Policy Name** fields, respectively.

> **NOTE:** If you are using these fields, then ensure the policies are already created and assigned to the organization.

6. Set the **Additional Inputs**, such as locale information for the current transaction.

7. Click **Authenticate** to authenticate to the configured WebFort Server.

   After successful authentication, the Sample Application returns credential details, transaction ID, notes, and authentication token.

Fetching User-Password Details

To fetch the User-Password details for the specified user:

1. Under the **User Password** sidebar menu, click the **Fetch** link.

   The Fetch User-Password page appears.

2. Enter the **User Name** who you want to authenticate using the User-Password mechanism.

3. Optionally, specify the **Organization** for the user.

4. Set the **Additional Inputs**, such as locale information for the current transaction.

5. Click **Fetch** to retrieve the User-Password details.

Glossary

Adobe Signature Service Protocol	See ASSP.
Administration Console	Web-based console for configuring communication mode between WebFort Server and its components and for performing administrative tasks.
ArcotID	Is a secure software credential that allows hardware level authentication in software form.
ASSP	Allows Acrobat and Reader users to access their roaming credentials for digital signatures. ASSP passes the hash to an ASSP-enabled server for signature and then after signing, embeds it into the end user's document.
Authentication	Is a process by which an entity proves that it is who it claims to be.
Authentication Policy	Set of rules that control the authentication process.
Authentication SDK	APIs that can be invoked by your application to forward authentication requests to WebFort Server.
Authentication Token	A token is an object that an authorized user of computer services is given to aid in authentication.
Credential	A proof of user identity. Digital credentials might be stored on hardware such as smart cards or USB tokens or on the server. They are verified during authentication.
Credential Profile	Common, ready-to-use credential configuration that can be shared among multiple organizations and multiple users.
Cryptographic Hash Function	A cryptographic hash function is a hash function with additional security properties, used in security-related applications such as authentication.
Default Organization	The Organization created by default when you deploy the Administration Console.
Digest-MD5	Is a widely used cryptographic hash function with a 128-bit hash value.

Digital Certificates	A certificate is a digital document that vouches for the identity and key ownership of an individual, a computer system, or an organization. This authentication method is based on the PKI cryptography method.
Encryption	The process of scrambling information in a way that disguises its meaning.
Error Message	Message returned by application to report to the user agent regarding any erroneous situations.
Forgot Your Password (FYP)	If the user forgets his ArcotID password, then a QnA session is carried out between the User and WebFort. On answering a minimal set of questions, the user is asked for a new ArcotID password and a new ArcotID is issued.
Instance	A system where WebFort Server is available at a specified port.
Issuance SDK	APIs that can be invoked by your application to forward issuance requests to WebFort Server for enrolling users and for creating their credentials in WebFort.
N-Strikes	The maximum number of failed authentication attempts that can be made by the user before locking out.
One-Time Password	Password credential valid for a single session. WebFort provides multi-use OTPs.
One-Time Token	Token returned by WebFort Server after successful authentications.
Organization	A WebFort unit that can either map to a complete enterprise (or a company) or a specific division, department, or other entities within the enterprise.
OTP	See One-Time Password.
OTT	See One-Time Token.
PKCS	PKCS refers to a group of Public Key Cryptography Standards devised and published by RSA. See Public-key Cryptography for more details.
PKCS#12	Defines a file format commonly used to store private keys with accompanying Public key certificates protected with a password-based symmetric key.
Private Key	One of a pair of keys used in public-key cryptography. The private key is kept secret and can be used to decrypt/encrypt data.
Public Key	One of a pair of keys used in public-key cryptography. The public key is distributed freely and published as part of a certificate. It is typically used to encrypt data sent to the public key's owner, who then decrypts the data using the corresponding private key.
Public Key Infrastructure (PKI)	The standards and services that facilitate the use of public-key cryptography and certificates in a networked environment.

Public-key Cryptography	Public-key cryptography is a form of modern cryptography which allows users to communicate securely without previously agreeing on a shared secret. Unlike symmetric cryptography, it uses two keys -- a public key known to everyone and a private or secret key known only to the owner of the public and private key pair. Public key cryptography is also called asymmetric cryptography.
QnA	A challenge-response authentication mechanism, QnA allows for a back and forth dialog between the user agent and server, where the server asks arbitrary number of questions, and the user supplies correct answers.
Questions and Answers	See QnA.
RADIUS Remote Authentication Dial In User Service	Protocol for centralized Authentication, Authorization, and Accounting (AAA).
SAML	XML standard for exchanging authentication data between an identity provider (provides assertions) and a service provider (uses assertions).
Sample Application	Demonstrates the usage of WebFort Java APIs and how your application can be integrated with WebFort. It can also be used to verify if WebFort was installed successfully, and if is able to perform issuance and authentication operations.
Secure Hash Algorithm (SHA)	Secure Hash Algorithm (SHA) family is a set of cryptographic hash functions.
Security Assertion Markup Language	See SAML
Single Sign-On (SSO)	SSO refers to a single identity that is shared across multiple systems. SSO lets a user logon once to a computer or network and access multiple applications and systems using a single credential.
Secure Sockets Layer (SSL)	SSL is a protocol intended to secure and authenticate communications across public networks by using data encryption.
User Name-Password	One of the credentials issued to the user during enrollment.
WebFort	Strong authentication system for authenticating end users.
WebFort Server	Server component that communicates with and accepts issuance and authentication requests from your application through WebFort SDKs.

AWF01-004DC-06000

Index

Index