CA ArcotID OTP Platform Independent Java Library

Authentication Developer's Guide

r2.2

Contact CA Technologies

Contact CA Support

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

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

Providing Feedback About Product Documentation

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

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

Contents

Chapter 1: Introduction

Computers and mobile devices are now also used as a medium for home banking and performing financial transactions. Because these transactions involve sensitive user data, relying on just user name for authentication is not sufficient.

To secure the online transactions from Man-in-the-Middle (MITM) and other related attacks, CA AuthMinder provides client applications that are based on CA ArcotID PKI and CA ArcotID OTP credentials. These software credentials provide two-factor authentication and are based on the patented Cryptographic Camouflage technique for securely storing keys.

There are client applications for computers and mobile devices.

To address your business requirements, you can extend and use the ArcotID OTP Client application. Alternatively, you can develop your own client application by using the Software Development Kit (SDK) that is shipped along with the client application.

This guide is intended to be a reference manual for you as you enhance the existing client application or create custom client applications for use with the ArcotID OTP for authentication on computers and mobile devices.

Note: The platform-independent version of the ArcotID OTP SDK is written in Java. This guide provides information about this platform-independent Java library (later referred to as "SDK" in this guide). For information about the ArcotID OTP SDK that is written in the Java, JavaScript, and Objective C programming languages for various mobile devices and computers, see the *CA ArcotID OTP Authentication Developer's Guide*.

Important! Implementations of most client features provided in the SDK can be used with very little or no modification. However, only *sample* implementations of the Device Locking feature and the Storage feature are provided in the SDK. If you want to use these two features, you must use the SDK to develop your own implementations.

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

ArcotID OTP Overview

ArcotID OTP is a One-Time Password compliant to OATH standards. The client application that you build by using ArcotID OTP SDK takes the user's PIN as an input and generates passwords (also known as *passcodes*) on the user's device. The user uses this generated passcode at the Web application or at the VPN gateway that is protected by ArcotID OTP authentication. Based on the authentication result, the user is granted access to the protected application.

The passcode generation is an offline process, which means the client application need not connect to the authentication server for generating passcodes.

ArcotID OTP library supports industry-standard passcode generation methods such as, counter-based passwords (HOTP), time-based passwords (TOTP), MasterCard Chip Authentication Program (CAP), and VISA Dynamic Passcode Authentication (DPA).

Chapter 2: Preparing for Integration

Before you start writing your code to integrate your application with ArcotID OTP SDK, ensure that:

- A release of CA AuthMinder that is supported by this release of ArcotID OTP Client is installed and running on a publically accessible system.
- The ArcotID OTP profile is set by using the Administration Console.

  Note: The lifecycle management of ArcotID OTP credential is handled by AuthMinder. By default, these credentials have default settings, which will be used during issuance. If you want to change these settings, then configure the credential profiles by using Administration Console. Refer to *CA AuthMinder Administration Guide* for more information.
- The application you are integrating with supports the operating systems listed in the "Integration Requirements (see page 9)" section.

Integration Requirements

The following table lists the software required to integrate the ArcotID OTP platform-independent Java library with your application.

Operating System
Any operating system that runs Java Runtime Environment (JRE) 6.0 or later.

Chapter 3: Understanding ArcotID OTP APIs

This chapter discusses the ArcotID OTP Software Development Kit (SDK) that you can use to build client applications for authenticating users by using their ArcotID OTP. The most common tasks performed using this SDK are **provisioning the ArcotID OTP account** to the user's device and **generating passcodes**. Other tasks that you can perform using the SDK include resetting ArcotID OTP PIN, fetching and deleting the accounts from default location, storing accounts in a custom location, locking account to the device by using device parameters of your choice, and checking library version.

The chapter first introduces you to the interfaces and classes that you will be using for different tasks mentioned in the preceding paragraph, and later explains the usage in detail.

- **Provisioning (Downloading) ArcotID OTP Accounts**

  To perform ArcotID OTP authentication, you need to first create an account for the user that contains the ArcotID OTP information and save it on their device. The Provisioning ArcotID OTP Accounts (see page 12) section discusses the provisionRequest() method in the OTP class that you need to use to create ArcotID OTP accounts.

- **Storing Accounts**

  After an account has been provisioned, it must be stored on the user's device. The Choosing Custom Storage Medium (see page 16) section provides information about storing accounts. The Storing Accounts in Memory (see page 17) section describes a sample implementation for using memory as the storage location.

- **Generating Passcodes**

  To perform ArcotID OTP authentication, the users have to first generate passcode, which is required during authentication. The Generating Passcodes (see page 17) section discusses the generateOTP() method in OTP class that you need to use to generate passcodes.

- **Resetting ArcotID OTP PIN**

  The Resetting ArcotID OTP PIN (see page 20) section discusses the resetPin method in the OTP class that you can use to change the user's ArcotID OTP PIN.

- **Managing Accounts**

  The Managing Accounts (see page 20) section discusses the methods of OTP class that you need to use for reading and deleting ArcotID OTP accounts stored in the default location.

- **Device Locking**

  Depending on the device that is being used, ArcotID OTP library supports default parameters for locking the account to the device. If you want to lock an account to the device by using the device parameters of your choice, then implement the DeviceLock interface, as discussed in the Device Locking (see page 23) section.

- **Reading ArcotID OTP Account Details**

  The Reading ArcotID OTP Account Details (see page 24) section discusses the OTP class fields that hold the ArcotID OTP details such as, unique identifier for the account, timestamp when the account was used, number of times the account was used, and friendly name for the account. It also discusses the classes that are used to set and get additional ArcotID OTP attributes.

- **Synchronizing the Client and Server**

  The Synchronizing the Client and Server (see page 27) section discusses how to use the syncRequest() method to construct and send a synchronization request to the server.

- **Checking ArcotID OTP SDK Version**

  The Checking Library Version (see page 29) section discusses the getVersion() method in the OTP class for checking the version of the ArcotID OTP SDK.

- **Converting the ArcotID OTP**

  The Converting the ArcotID OTP (see page 29) section discusses the provisionRequest() method and callback() method for converting the ArcotID OTP into a string and back into an account object.

Provisioning ArcotID OTP Accounts

The provisionRequest() method is used to construct and send an account provisioning request to the server. The callback() method is invoked after the request is sent to the server. In an asynchronous implementation, typically, the provisionRequest() method and callback() method are invoked multiple times until the success status is received in the callback() method.

Note: The provisionRequest() method replaces the provisionAccount() method in the SDK. The provisionAccount() method has been retained in the SDK only for backward compatibility purposes.

API Details

The following table lists the input and output parameters of the provisionRequest() method:

Parameter	Description
Input Parameters	
args	Map containing name-value pairs. For the initial provisionRequest call, the following name-value pairs are populated in the map: URL: Request URL string ACCOUNTID: User identifier string ACTCODE: Activation code to validate the request. CONN_OBJECT: (Optional) Network connection object Refer to JavaDocs for more information on the other supported name-value pairs.
Output Parameters	
None.	

How a Provisioning Request Works

This section explains the sequence of events that take place when a provisioning request is sent to the server. Apply this information while using the provisionRequest() method and callback() method in your implementation.

The following events take place when a provisioning request is sent to the server:

1. When the provisionRequest() method is invoked the first time, it sends the provisioning request to the server. This request contains the following parameters:
   - URL
   - ACCOUNTID
   - ACTCODE
   - CONN_OBJECT (optional)
2. The callback() method is invoked. The parameters returned by this method depend on what happens next:

   Note: The client application must free up the parameters map after each provisionRequest() call and the corresponding callback() call.

   - If the request succeeds, then the callback() method returns the following parameter values:
     - URL
     - ACCOUNTID
     - ACTCODE
     - CONN_OBJECT (only if this parameter was sent in the initial request)
     - REQUESTTYPE=”provisioning”
     - PINTYPE=1/0
     - MINPINLENGTH=*<minimum-PIN-length>*
     - XML=*<response>*
     - DLTA=*<time-difference-between-the-server-and-client-application>*
     - STATE=PINREQUIRED

       The PINREQUIRED state indicates that the server is requesting the PIN.
   - If the request fails for any reason, then the callback() method returns the following parameter values:
     - URL
     - ACCOUNTID
     - ACTCODE
     - CONN_OBJECT (only if this parameter was sent in the initial request)

- REQUESTTYPE=”provisioning”
- ERR_CODE=<*error-code*>
- ERR_MSG=<*error-message*>
- PLATFORM_MSG=<*error-message-returnedf-by-the-underlying-platform*>

3. The provisionRequest() method is invoked again. The parameters passed by this method depend on the state returned by the callback() method. For the PINREQUIRED state of the callback() method, the following are the parameters (with some sample values) sent by the provisionRequest() method. Note that the original set of parameters are included in the request again.
   - URL
   - ACCOUNTID
   - ACTCODE
   - CONN_OBJECT (only if this parameter was sent in the initial request)
   - REQUESTTYPE=”provisioning”
   - PINTYPE
   - MINPINLENGTH
   - XML
   - DLTA
   - STATE=PINREQUIRED
   - PINVALUE
4. After the request is processed by the server, the parameters returned by the callback() method depend on whether the request has succeeded or failed:
   - If the request was successfully processed, then the callback() method returns the following parameter values:
     - URL
     - ACCOUNTID
     - ACTCODE
     - CONN_OBJECT (only if this parameter was sent in the initial request)
     - REQUESTTYPE=\xE2\x80\x9Dprovisioning”
     - ACCOUNT_KEY=<*newly-created-account-key*>
     - PINVALUE=<*masked-newly-created-PIN*>
     - STATE=DONE
   - If the request fails, then the callback() method returns the following parameter values:
     - URL

- ACCOUNTID
- ACTCODE
- CONN_OBJECT (only if this parameter was sent in the initial request)
- REQUESTTYPE=”provisioning”
- ERR_CODE=*<error-code>*
- ERR_MSG=*<error-message>*
- PLATFORM_MSG=*<error-message-returned-by-the-underlying-platform>*

Exception

ArcotOTPCommException is returned if any errors are encountered while processing a provisioning request. See chapter, "ArcotID OTP SDK Exceptions and Error Codes" (see page 31) for more information on the exception class and errors returned by ArcotID OTP SDK.

Choosing Custom Storage Medium

ArcotID OTP library enables you to store accounts in the storage medium of your choice. You implement the Store interface to define the storage medium, and then set that as the default.

Perform the following steps to set up a custom storage:

1. Implement the Store interface to use the custom storage.
2. Invoke the setStore() method in the OTP class to initialize the storage medium.

The ProxyStore class is a basic implementation of the Store interface. Each method of this class throws RuntimeException to indicate that it is just a sample and that it must be replaced.

Storing Accounts in Memory

ArcotID OTP library provides a sample implementation for storing the accounts in device memory.

Note:

- If you exit the application, then the data stored in the memory will be lost.
- The sample implementations that are provided with the ArcotID OTP library must be used for reference only.

Perform the following steps to store accounts in memory:

1. Invoke the MemoryStore class to use memory as a storage medium.
2. Invoke the setStore() method in the OTP class to initialize the storage medium.

Generating Passcodes

To perform ArcotID OTP authentication, users have to first generate a passcode on their device and then submit it at the authenticating website to access the protected source. To generate the passcode, use the generateOTP() method in the API class.

ArcotID OTP SDK supports major industry-standard One-Time Password (OTP) generation algorithms. Based on the algorithm that you are using, you must prepare the input data and hash this data. The following table lists the fields that hold the input data required for generating passcodes.

Note: MasterCard Chip Authentication Program (CAP) and VISA Dynamic Passcode Authentication (DPA) algorithms support different modes for generating passcodes. Refer to the vendor documentation for more information on these modes.

Field	Description
CAP and DPA Password Fields	
P_MODE	Specifies that modes are being used for generating passcodes. The possible values for this parameter are: ■ M_1 ■ M_2 ■ M_2_TDS ■ M_3

Field	Description
M_1	Specifies that Mode 1 is being used for generating passcodes. If you are using this mode, then you have to collect the following information: ■ P_AA ■ P_TRCC ■ P_UN
M_2	Specifies that Mode 2 is being used for generating passcodes. This mode does *not* require any other additional information.
M_2_TDS	Specifies that Mode 2 with TDS is being used for generating passcodes. This mode supports10 entries for passing any additional information.
M_3	Specifies that Mode 3 is being used for generating passcodes. If you are using this mode, then you have to collect P_UN.
P_AA	Specifies the amount that is used in the transaction.
P_TRCC	Specifies the type of currency that is used in the transaction.
P_UN	Specifies the challenge that is used in the transaction.
P_DATA	This field is used to pass additional information that is required by M_2_TDS mode.
Time-Based OTP Password Fields	
P_TIME	Specifies the time for which the OTP is valid. **Note:** This value needs to be provided only if an OTP is being generated for a time other than the current time.

API Details

The following table lists the input and output parameters of the generateOTP() method.

Parameter	Description
Input Parameters	
Id	The unique identifier of the account.
pwd	ArcotID OTP PIN.

Parameter	Description

params	The parameters that are required for generating passcodes. You need to set the parameters based on the type of OTP to be generated. For example:
	
	■ TOTP
	Hashtable params = new Hashtable();
	params.put(OTP.P_TIME, "123456789");
	Note: As mentioned in the previous table, the P_TIME value needs to be provided only if an OTP is being generated for a time other than the current time.
	
	■ CAP or DPA Mode 1
	Hashtable params = new Hashtable();
	params.put(OTP.P_MODE, OTP.M_1);
	params.put(OTP.P_AA, "123.45");
	params.put(OTP.P_UN, "0123456789");
	
	■ CAP or DPA Mode 2 with TDS
	Hashtable params = new Hashtable();
	params.put(OTP.P_MODE, OTP.M_2_TDS);
	params.put(OTP.P_DATA + "0", "123");
	params.put(OTP.P_DATA + "1", "456");
	params.put(OTP.P_DATA + "2", "789");
	
	■ CAP or DPA Mode 3
	Hashtable params = new Hashtable();
	params.put(OTP.P_MODE, OTP.M_3);
	params.put(OTP.P_UN, "0123456789");

Output Parameters	
The generated passcode.	

Exception

The OTPException class is returned if there any errors while signing the challenge. See chapter, "ArcotID OTP SDK Exceptions and Error Codes" (see page 31) for more information on the exception class and errors returned by ArcotID OTP SDK.

Resetting ArcotID OTP PIN

The ArcotID OTP SDK provides functions that you can use to reset the user's ArcotID OTP PIN. The user might be prompted to reset their PIN, if they forget their current PIN. Before resetting the PIN, you must prompt the users need to perform secondary authentication to prove their identity. Typically, Security Questions and Answers or One-Time Passwords are used as secondary authentication mechanisms.

To reset the PIN, you need to use the resetPin() method in the OTP class.

API Details

The following table lists the input and output parameters of the resetPin() method.

Parameter	Description
Input Parameters	
id	The unique identifier of the account.
oldPin	User's current ArcotID OTP PIN.
newPin	New PIN that the user wants to set.
Output Parameters	
None.	

Exception

The OTPException class is returned if there any errors while executing the provisionAccount() method. See chapter, "ArcotID OTP SDK Exceptions and Error Codes" (see page 31) for more information on the exception class and errors returned by ArcotID OTP SDK.

Managing Accounts

This section discusses the APIs that you need to use for managing the accounts in default storage:

- Fetching Accounts (see page 21)
- Deleting Accounts (see page 22)

Fetching Accounts

To fetch the accounts from the default storage, you need to use the API class. This class provides different options to read accounts as mentioned in the following table:

Method	Description
getAccount()	Fetches the account based on the account identifier that is passed as an input.
getAllAccounts()	Fetches all the accounts that are present on the device. **Note:** You can also fetch accounts based on the ArcotID OTP namespace. To do this, pass the namespace as an input parameter to the getAllAccounts() method. This method fetches all the accounts whose domains match the namespace passed to the method. For example, if you pass ARCOT.COM as a namespace to the method, then it returns accounts belonging to ARCOT.COM, A.ARCOT.COM, B.ARCOT.COM, and so on.

API Details

The following table lists the input and output parameters of the getAccount() method:

Parameter	Description
Input Parameters	
id	The unique identifier of the account that has to be fetched.
Output Parameters	
account	The requested account.

The following table lists the input and output parameters of the getAllAccounts() method:

Parameter	Description
Input Parameters	
None.	
Output Parameters	
account	An array of all the accounts present in the storage.

The following table lists the input and output parameters of the getAllAccounts() method when a namespace is passed as input:

Parameter	Description
Input Parameters	
ns	The namespace of the requested accounts.
Output Parameters	
account	Array of accounts belonging to the specified namespace (domain) and also accounts from other namespaces that contain the search string in their name.

Exception

The OTPException class is returned if there any errors while reading the account from the storage location. See chapter, "ArcotID OTP SDK Exceptions and Error Codes" (see page 31) for more information on the exception class and errors returned by ArcotID OTP SDK.

Deleting Accounts

To delete accounts, use the deleteAccount() method in the API class.

API Details

The following table lists the input and output parameters of the deleteAccount() method:

Parameter	Description
Input Parameters	
id	The unique identifier of the account that has to be deleted.
Output Parameters	
None.	

Exception

The OTPException class is returned if there any errors while deleting the account from the storage location. See chapter, "ArcotID OTP SDK Exceptions and Error Codes" (see page 31) for more information on the exception class and errors returned by ArcotID OTP SDK.

Device Locking

Device locking enables an account to be locked to a specific device, so that the account is unusable if it is copied to another account to be locked to a specific device. A device is locked at the time when an account is stored on the device. By default, this feature is enabled.

Based on the operating system, ArcotID OTP SDK supports different parameters to derive the unique identifier of the device for locking the account. If you want to use other device parameters for device locking, then see Device Locking Using Non-Default Parameters (see page 23) for more information.

The SystemDeviceLock class is a basic implementation of the DeviceLock interface. This class is an extension of the ProxyDeviceLock class, and it is intended for use as a sample. To use the Device Locking feature, extend the ProxyDeviceLock class.

Note: You can disable the Device Locking feature by passing a NULL value to the setDeviceLock() method.

Device Locking Using Non-Default Parameters

To lock an account to a device by using attributes other than the default attributes supported by ArcotID OTP SDK:

1. Implement the getKey() method in the DeviceLock interface.

   The getKey() method returns the unique identifier of the device that you have requested.

2. Invoke the setDeviceLock() method in the OTP class.

   The setDeviceLock() method locks the account to the device by using the parameters that are fetched by the getKey() method.

API Details

The following table lists the input and output parameters of getKey() method:

Parameter	Description
Input Parameters	
None.	
Output Parameters	
device identifier	The unique identifier of the device.

Exception

The OTPException class is returned if there any errors while locking the account to the device See chapter, "ArcotID OTP SDK Exceptions and Error Codes" (see page 31) for more information on the exception class and errors returned by ArcotID OTP SDK.

Reading ArcotID OTP Account Details

This section walks you through the following topics related to Account class:

ArcotID OTP Details

The following table lists the fields that hold the basic ArcotID OTP information:

Field	Description
A_DLTA	The time difference between the client and the server.
A_IAF_AA	The attribute that specifies whether the amount is required to perform transaction.
A_IAF_TRCC	The attribute that specifies whether currency is required to perform transaction.
A_IAF_UN	The attribute that specifies whether challenge is required to perform transaction.
A_MPL	The attribute that holds the Minimum PIN Length of the account.
A_PINTYPE_ALPHANUMERIC	The attribute that specifies the possible PIN types for the account.
A_PINTYPE_NUMERIC	The attribute that specifies the possible PIN types for the account. A_PINTYPE_NUMERIC and A_PINTYPE_ALPHANUMERIC attributes are the possible values returned by getPINType().
A_PROTOCOLVER	The version of the protocol that is used for communication between the client and server.
A_RESETSUPPORT	The attribute for specifying whether reset is supported.

Field	Description
accountId	The unique identifier of the ArcotID OTP account.
algo	The algorithm that is used to generate passcodes.
creationTime	The timestamp when the ArcotID OTP account was created.
expiryTime	The timestamp when the ArcotID OTP account expires.
lastUsed	The timestamp when the ArcotID OTP account was last used.
logoUrl	The URL of the logo image, this image is displayed on the application. **Note:** This field is for future use.
name	A user friendly name for the account.
ns	The namespace of the ArcotID OTP. It is typically the domain name to which the ArcotID OTP belongs to.
org	The organization to which the user for whom the account is being created belongs.
provisioningURL	This field is deprecated. Use the provUrl field.
provUrl	The URL of the AuthMinder authentication server.
uses	The number of times ArcotID OTP has been used.

Fetching ArcotID OTP Details

The following table lists the input and output parameters of the getId() method that is used to fetch the ArcotID OTP information:

Parameter	Description
Input Parameters	
None.	
Output Parameters	
Identifier of the ArcotID OTP account.	

Managing Additional ArcotID OTP Attributes

To set the ArcotID OTP information that cannot be passed by using the fields listed in ArcotID OTP Details (see page 24), you need to use the setAttribute() method and pass that information as name-value pairs, and to read that information use getAttribute() method. The following table explains these methods:

Method	Description
setAttribute()	This method is used to set the ArcotID OTP information that cannot be passed by using the fields listed in ArcotID OTP Details (see page 24). The additional information is passed as name-value pairs. **Input Parameters:** ■ The name of the attribute. For example, if you want to display your organization copyright information along with the user details on the application, then you can set a new attribute called *Copyright*. **Output Parameters:** ■ The value (in string format) that has to be set for the attribute.
getAttribute()	This method is used to read the value of ArcotID OTP attributes. **Input Parameters:** ■ The name of the attribute, whose value has to be fetched. **Output Parameters:** ■ Attribute value in string format.
getMinPINLength()	This method is used to get the minimum PIN length that must be used for the account.
getPINType()	This method is used to get the PIN type of the account. A_PINTYPE_ALPHA_NUMERIC and A_PINTYPE_NUMERIC are examples of the PIN types.

Saving Additional ArcotID OTP Attributes

After you set a new ArcotID OTP attribute or change any existing ArcotID OTP attribute, you need to save these changes by invoking the saveAccount() method in the API class.

Perform the following steps to save the changes made to accounts:

1. Prepare an instance of the Account object that has to be saved.
2. Invoke the saveAccount() method to save the account.

API Details

The following table lists the input and output parameters of the saveAccount() method:

Parameter	Description
Input Parameters	
acc	The account that has to be saved.
Output Parameters	
None.	

Exception

The OTPException class is returned if there any errors while checking the library version. See chapter, "ArcotID OTP SDK Exceptions and Error Codes" (see page 31) for more information on the exception class and errors returned by ArcotID OTP SDK.

Synchronizing the Client and Server

The ArcotID OTP account details on the client must be in synchronization with the server. Otherwise, authentication will fail. If the account details are not in synchronization, then use the syncRequest() method in the Account class to re-synchronize them. The syncRequest() method is used to construct and send a synchronization request for fetching the OTP counter or timestamp synchronization value from the server. If the request is successful, the client receives the success status in the callback() method.

API Details

The following table lists the input and output parameters of the syncRequest() method:

Parameter	Description
Input Parameters	
args	Map containing the following name-value pairs: ACCOUNT: Account object CONN_OBJECT: (Optional) Network connection object
Output Parameters	
None.	

How a Synchronization Request Works

This section explains the sequence of events that take place when a synchronization request is sent to the server. Apply this information while using the syncRequest() method and callback() method in your implementation.

The following events take place when a synchronization request is sent to the server:

1. When the syncRequest() method is invoked the first time, it sends the synchronization request to the server. This request contains the following parameters:
   - ACCOUNT_OBJ
   - CONN_OBJECT (optional)
2. The callback() method is invoked. The parameters returned by this method depend on what happens next:

   Note: The client application must free up the parameters map after each provisionRequest() call and the corresponding callback() call.

   - If the request succeeds, then the callback() method returns the following parameter values:
     - REQUESTTYPE =”softsync”
     - ACCOUNT_OBJ
     - CONN_OBJECT (only if this parameter was sent in the initial request)
     - STATE=DONE
   - If the request fails, then the callback() method returns the following parameter values:
     - REQUESTTYPE=”softsync”
     - ERR_CODE=*<error-code>*
     - ERR_MSG=*<error-message>*
     - PLATFORM_MSG=*<error-message-returned-by-the-underlying-platform>*

Exception

ArcotOTPCommException is returned if any errors are encountered while sending a synchronization request. See chapter, "ArcotID OTP SDK Exceptions and Error Codes" (see page 31) for more information on the exception class and errors returned by ArcotID OTP SDK.

Fetching Library Version

To fetch the version of the ArcotID OTP library that you are using, you need to use the getVersion() method in the API class.

API Details

The following table lists the input and output parameters of the getVersion() method:

Parameter	Description
Input Parameters	
None.	
Output Parameters	
The version number of the ArcotID OTP library.	

Exception

The OTPException class is returned if there any errors while checking the library version. See chapter, "ArcotID OTP SDK Exceptions and Error Codes" (see page 31) for more information on the exception class and errors returned by ArcotID OTP SDK.

Serializing and Deserializing an Account Object

The AccountFormat class provides methods for converting an ArcotID OTP account object into a string and for converting the string representation of an ArcotID OTP into an account object.

API Details

The parse() method converts the string representation of an ArcotID OTP into an account object. The following table lists the input and output parameters of the parse() method:

Parameter	Description
Input Parameters	
s	String representation of an ArcotID OTP account object.

Parameter	Description
Output Parameters	
ArcotID OTP account object.	

The format() method converts an ArcotID OTP account object into a string. The following table lists the input and output parameters of the format() method:

Parameter	Description
Input Parameters	
Account	ArcotID OTP account object.
Output Parameters	
String representation of the ArcotID OTP account object.	

Chapter 4: ArcotID OTP SDK Exceptions and Error Codes

This chapter lists all exceptions and error codes that are returned by ArcotID OTP SDK. It covers the following topics:

Exceptions

If there are any errors while processing the ArcotID OTP APIs, then the OTPException class is returned. This class provide an constructor class OTPException, which takes error code, error message, and throwable as input.

To fetch the error code for a particular error, the OTPException class provides getcode() method, which returns the error code.

Error Codes

The following table lists the error codes returned by ArcotID OTP APIs:

Code	Code Message	Description
Default Errors		
1	E_UNKNOWN	Internal error.
Storage Errors (10-19)		
11	E_STORE_WRITE	There was an error while saving the account.
12	E_STORE_READ	There was an error while reading the account.
13	E_STORE_DELETE	There was an error while deleting the account.
14	E_STORE_ACCESS	There was an error while accessing the account.
User Input Errors		
31	E_BAD_NS	The One-Time Password (OTP) algorithm is invalid.
32	E_BAD_XML	The Activation Code provided by the user is invalid.
33	E_BAD_ID	The user identifier is invalid.

Code	Code Message	Description
34	E_BAD_ACCOUNT	The URL of AuthMinder Server is not configured correctly.
35	E_BAD_PIN	The ArcotID OTP passcode entered by the user is invalid.
36	E_BAD_ALGO	The algorithm used for generating passcode is invalid.
37	E_BAD_CS	The ArcotID OTP card string is invalid.
38	E_BAD_ATTR	The attributes of the ArcotID OTP card passed are invalid.
Processing Errors		
41	E_PROC_SERVER	The AuthMinder Server returned an error.
42	E_PROC_XML	There was an error while processing the input XML.
43	E_PROC_DEVLOCK	There was an error while locking the ArcotID OTP to the device.
ArcotID OTP Card Errors		
51	E_TOTP_TIME	The time period for which the TOTP is valid has elapsed.
52	E_CAP_MODE	The Client Authentication Program (CAP) mode used for generating passcode is invalid.
53	E_CAP_AA	The amount key is either not passed or the key length is incorrect.
54	E_CAP_TDS	The key to specify Mode2TDS mode is either not passed or the key length is incorrect.
55	E_CAP_TRCC	The currency challenge key is either not passed or the key length is incorrect.
56	E_CAP_UN	The challenge key is either not passed or the key length is incorrect.