RSA® Authentication Agent API 8.5 for C Developer's Guide

Revision 1

Contact Information

Go to the RSA corporate website for regional Customer Support telephone and fax numbers:
www.emc.com/domains/rsa/index.htm

Trademarks

RSA, the RSA Logo and EMC are either registered trademarks or trademarks of EMC Corporation in the United States and/or other countries. All other trademarks used herein are the property of their respective owners. For a list of EMC trademarks, go to **www.emc.com/legal/emc-corporation-trademarks.htm#rsa**.

License Agreement

This software and the associated documentation are proprietary and confidential to EMC, are furnished under license, and may be used and copied only in accordance with the terms of such license and with the inclusion of the copyright notice below. This software and the documentation, and any copies thereof, may not be provided or otherwise made available to any other person.

No title to or ownership of the software or documentation or any intellectual property rights thereto is hereby transferred. Any unauthorized use or reproduction of this software and the documentation may be subject to civil and/or criminal liability. This software is subject to change without notice and should not be construed as a commitment by EMC.

Third-Party Licenses

This product may include software developed by parties other than RSA. The text of the license agreements applicable to third-party software in this product may be viewed in the folder Third-Party Licenses on the DVD**.**

Note on Encryption Technologies

This product may contain encryption technology. Many countries prohibit or restrict the use, import, or export of encryption technologies, and current use, import, and export regulations should be followed when using, importing or exporting this product.

Distribution

Use, copying, and distribution of any EMC software described in this publication requires an applicable software license.

EMC believes the information in this publication is accurate as of its publication date. The information is subject to change without notice.

THE INFORMATION IN THIS PUBLICATION IS PROVIDED "AS IS." EMC CORPORATION MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND WITH RESPECT TO THE INFORMATION IN THIS PUBLICATION, AND SPECIFICALLY DISCLAIMS IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

March, 2014
Revised: May 2015

Contents

Revision History

Revision Number	Date	Revision
1	May 2015	Updated the instructions for "Using Dynamic Libraries or Static Libraries in UNIX."

Preface

About this Guide

This guide describes the RSA Authentication Agent application programming interfaces (APIs), and how to use them. This guide is intended for developers who use these APIs to integrate RSA SecurID into custom or third-party applications.

RSA recommends that developers using the Authentication Agent API have an understanding of both synchronous and asynchronous function calls, callback mechanisms, and the implications of designing and developing multithreaded code.

Developers using this API to create their own agents must also have an understanding of the basic operations involved in RSA SecurID authentication, which include:

- Verifying the user's passcode
- Processing the data when a user creates a new PIN in New PIN mode
- Verifying the tokencode when a user is placed in Next Tokencode mode

Developers must also take necessary precautions to protect all user-supplied data when in use, and clear the data when it is not in use.

RSA Authentication Agent API Documentation

For more information about RSA Authentication Agent API, see the following documentation:

Release Notes. Provides information about what is new and changed in this release, as well as workarounds for known issues. The latest version of the *Release Notes* is available on RSA SecurCare Online at **https://knowledge.rsasecurity.com**.

RSA Authentication Agent API 8.5 for C Developer’s Guide. Provides information about developing custom programs using the RSA Authentication Agent API. Includes an overview of the API features as well as detailed descriptions of the API functions.

Related Documentation

For more information about products related to RSA Authentication Agent API, see the following:

RSA Authentication Manager documentation set. The full documentation set for RSA Authentication Manager is included in the documentation folder available with the Authentication Manager files.

Support and Service

RSA SecurCare Online	**https://knowledge.rsasecurity.com**
Customer Support Information	**www.emc.com/support/rsa/index.htm**
RSA Ready Partner Solution Gallery	**https://gallery.emc.com/community/marketplace/rsa**

RSA SecurCare Online offers a knowledgebase that contains answers to common questions and solutions to known problems. It also offers information on new releases, important technical news, and software downloads.

The RSA Ready Partner Solutions Gallery provides information about third-party hardware and software products that have been certified to work with RSA products. The directory includes Implementation Guides with step-by-step instructions about interoperation of RSA products with these third-party products.

Before You Call Customer Support

Make sure that you have direct access to the computer running the RSA Authentication Manager software.

Please have the following information available when you call:

- ❑ Your RSA Customer/License ID. You can find this number on the license distribution medium or by running the Configuration Management application on Windows platforms, or by typing **sdinfo** on UNIX platforms.
- ❑ RSA Authentication Agent API software version number.
- ❑ The make and model of the machine on which the problem occurs.
- ❑ The name and version of the operating system under which the problem occurs.
- ❑ The Authentication Manager server version number.
- ❑ Any error message shown or logged by the Authentication API or the Authentication Manager server.

1 Overview of the RSA Authentication Agent API

The RSA SecurID solution provides two-factor authentication to protect access to data and applications. This access can occur through remote dial-in connections, local access, domain and terminal services access, internet and VPN connections, intranet and extranet applications.

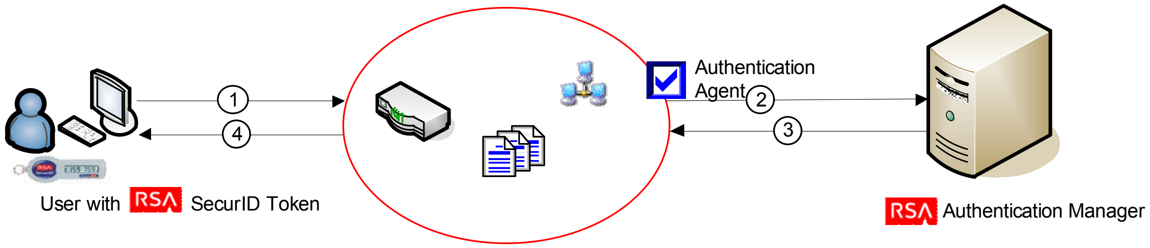
The SecurID solution consists of an Authentication Manager server, an authentication agent that communicates with it, and authenticators that provide the tokencode.

The authentication agent initiates a SecurID authentication session when a user attempts to access a protected resource. It verifies data provided by the user with the data stored in the Authentication Manager server. Based on the result, the user is either allowed or denied access.

RSA provides authentication agent APIs, which can be used to develop a custom agent to communicate with the Authentication Manager server. These APIs are available in both static and dynamic link libraries for Windows and UNIX platforms. Using these APIs, you can develop custom agents to protect required resources.

Typical Functioning of an Authentication Agent

An agent created using the RSA Authentication Agent APIs performs the steps detailed in the following figure in a secure manner.



1. Intercepts all access attempts, such as attempts to log on or access a URL.
2. Determines whether the requested resource is protected by RSA SecurID:
   - If the requested resource is not protected by RSA SecurID, the agent either ignores the request, or takes appropriate action, such as writing an audit message in the UNIX syslog or in the Windows Event Log.
   - If the requested resource is protected by RSA SecurID, the agent continues the authentication process.

3. Prompts the user for the user name so that RSA Authentication Manager can validate that the tokencode is generated from the authentication device registered to that user.
4. Locks the user name to prevent replay attacks when Authentication Manager replicas are deployed (Authentication Manager 6 only).
5. Requests the user for passcode.
6. Combines the passcode with a secret known only to the agent and its associated RSA Authentication Manager realm, and delivers the combined data to a server for validation:
   - If Authentication Manager approves the request, the agent grants access to the protected resource and takes appropriate action.
   - If Authentication Manager denies access, the agent denies access to the protected resource and takes appropriate action.

In addition to providing basic access checks during standard authentication, agents also handle several security-related housekeeping tasks, such as those related to Next Tokencode mode and New PIN mode. In Next Tokencode mode, the Authentication Manager requests the next tokencode displayed on the user's token. If the next tokencode is sent to the Authentication Manager and matches the expected tokencode, the authentication succeeds.

The Authentication Manager administrator determines if the user associated with a particular token requires a new PIN. The administrator also determines the characteristics of the PINs, which your custom agent can test using the API functions.

RSA Authentication Agent API 8.5 New and Updated Features

Unless specifically mentioned, the features are available on all supported platforms.

IPv6 Support

RSA Authentication SDK 8.5 supports authentication agents that use IPv4 or IPv6 addresses, and the HTTP and TCP protocols in addition to the traditional UDP protocol.

Backward Compatibility

After upgrading from SDK 8.1, the SDK 8.5 will not communicate with RSA Authentication Manager 6.1 and 7.1. However, if required, code that integrates with both the 8.1 SDK and the 8.5 SDK can be written to preserve backward compatibility.

For information about backward compatibility with older versions of the APIs, see "Backward Compatibility" on page 25.

AES 256 Cryptography Support

RSA Authentication SDK 8.5 supports AES 256 cryptography for protecting communication between the agent and the server. AES 256 provides improved confidentiality over RC5 128, which is supported in SDK 8.1 and in earlier versions.

Agent-Server Trust Model - Optional Node Secret

In SDK 8.5, a node secret is used for client authentication; it is optional. To enable client authentication, the node secret should be present for the agent on the Authentication Manager side. The same node secret should be downloaded and its path should be specified in **rsa_api.properties** using SDNDSCRT_LOC tag. In SDK 8.5, instead of a node secret, a dynamically negotiated key is used to encrypt the channel along with a strong encryption algorithm. There is no over-the-network mechanism to set the agent's secret key (or credential); it can only be set using the **agent_nsload** utility. This new trust mechanism is considered more secure and IP-agnostic.

This differs from SDK 8.1, where the trust between the server and agent is established and maintained using a node secret, a file that resides on the agent host. In SDK 8.1, the node secret is mapped to the IP (v4) address of the agent host, and is mandatory in UDP agents. Authentication agents use the node secret to encrypt authentication requests that they send to Authentication Manager. Authentication Manager automatically creates and sends the node secret to the agent in response to the first successful authentication on the agent.

For more information about node secret encryption, see the *RSA Authentication Manager 8.x Administrator's Guide.*

Agent Name

In RSA Authentication SDK 8.5, the agent can be configured with an agent name. Unlike the UDP agents, a TCP agent uses a logical name to identify agents. An agent name is not required to be a fully qualified host name and does not require an IP address. An agent name can be specific, such as a host name, or generic, such as *CorporateDesktop* or *Engineering Web Server.*

Agents running on different physical hosts can share a logical agent name. Multiple logically named agents can reside on a single physical host.

Support for EAP 32 and the Generic Credential API

Authentication Manager versions 6.1 and later support Extensible Authentication Protocol EAP 32 as defined in the 802.1x security standard.

The Authentication API includes support for generic user credentials. This means that the deployed API is extensible to all new credential types that Authentication Manager supports in the future, without requiring any updates.

For information about the API functions that make up the Generic Credential API, see "Generic Credential API Functions" on page 19.

Note: You can use only one type of credential during the course of an authentication. For example, you cannot switch from EAP 32 to standard RSA SecurID authentication during the course of a single authentication pass.

This feature requires certain Authentication Manager configuration settings. For more information, see the *RSA Authentication Manager 8.x Administrator's Guide*.

User Prompts

The agent handles all input and output associated with processing an authentication request. The agent prompts the user to enter the input data (for example, Enter passcode:) and displays status information based on the return values and result codes (for example, Access denied). These prompts are supplied by certain APIs. The functions AceStartAuth and AceContinueAuth supply prompts that the agent provides to the user. For information about these prompts, see Appendix C, "Modifying the Message Catalog."

Load Balancing

An agent that you link with the Authentication Agent API can communicate with multiple Authentication Managers in a realm. The API incorporates load-balancing routines to help the agent select the best Authentication Manager for communication.

Load balancing occurs automatically based on data that the agent gathers dynamically at runtime and data acquired from configuration files.

There are three types of load balancing: manual, dynamic (default), or round-robin.

Manual Load Balancing

With manual load balancing, you specify the Authentication Manager server that each authentication agent host uses. The agent directs requests to servers based on the priority that you manually assign to each server. In this manner, the authentication agent can direct authentication requests to a few Authentication Manager servers more frequently than the others. To specify manual load balancing, include the USESERVER statement in the **sdopts.rec** file and associate priority settings with each Authentication Manager server you specify for use. For more information, see "Configuring the sdopts.rec File" on page 39.

Dynamic Load Balancing

With dynamic load-balencing, the authentication agent directs requests to Authentication Manager servers based on their dynamic priority. Priority is determined by a server response-time-based evaluation; servers with faster response times have higher priority. For each request, the server with highest priority is tried before the server with the next-highest priority (failover server) is tried. As requests are sent to each server, response time is evaluated each time and the priority is dynamically updated.

Round-Robin Load Balancing

With round-robin load balancing, the authentication agent directs requests to each Authentication Manager server, as provided by DNS resolution, in sequence, with no assigned priority or preference. For each request, a connection attempt is made with the "next" server in the list. If the connection fails, the request goes to the next server, and so on, until it reaches the end of the list, at which point the request goes back to first server in the list ("round-robin" style). This process continues until either a server accepts a connection or the connection retry count is reached.

AVOID keyword

To exclude a server from dynamic or round-robin load balancing, use the AVOID keyword in the **sdopts.rec** file. This keyword prevents the specified Authentication Manager server from being used during load balancing. For more information, see "Configuring the sdopts.rec File" on page 39.

Expiration Facility and Cleanup Callbacks

The AceSetTimeout function automatically terminates any outstanding authentication request, while the calls AceCleanup and AceShutdown terminate outstanding authentication requests and shut down the library.

Thread Safety

Unless otherwise specified, all Authentication Agent API functions (excluding AceInitialize) are thread-safe, which means you can safely call them from multithreaded applications without program failure or data corruption.

AceInitialize Error Detection

An AceInitialize must be called before any other API function is called, and must be called only once.

AceInitialize initializes the worker threads and loads the mandatory **sdconf.rec** file, and optionally, the **sdopts.rec** file, into memory.

The AceInit, AceInitEx, AceStartAuth, SD_Init, and SD_InitEx functions can detect whether the AceInitialize function has already been called, and if necessary, call the function for your code. Although it is the developer's responsibility to ensure that AceInitialize is called first, these functions can avoid an error condition by detecting whether the AceInitialize function has already been called.

If your application calls the AceInit, AceInitEx, AceStartAuth, SD_Init, or the SD_InitEx functions in a multithreaded environment, and multiple automatic calls to AceInitialize occur simultaneously, these functions return an error.

Note: AceInitialize is not thread-safe. Therefore, if your application calls the AceInit, AceInitEx, AceStartAuth, SD_Init or SD_InitEx APIs at the same time, the result is undefined.

Data Encapsulation

In API versions 5.0 and earlier, you could directly access the contents of API-specific data structures. Now, instead of direct access, your code must obtain a handle to the data area, and then use that handle in the API functions that provide the content of specific elements within the data structure.

The following functions provide data that was previously available by directly reading API-specific data structures: AceGetAlphanumeric, AceGetMaxPinLen, AceGetMinPinLen, AceGetShell, AceGetSystemPin, AceGetTime, AceGetUserData, and AceGetUserSelectable.

Synchronous and Asynchronous Functions

In SDK 8.5, asynchronous functions are implemented as synchronous functions. This has implications for existing applications that contain logic requiring an asynchronous implementation.

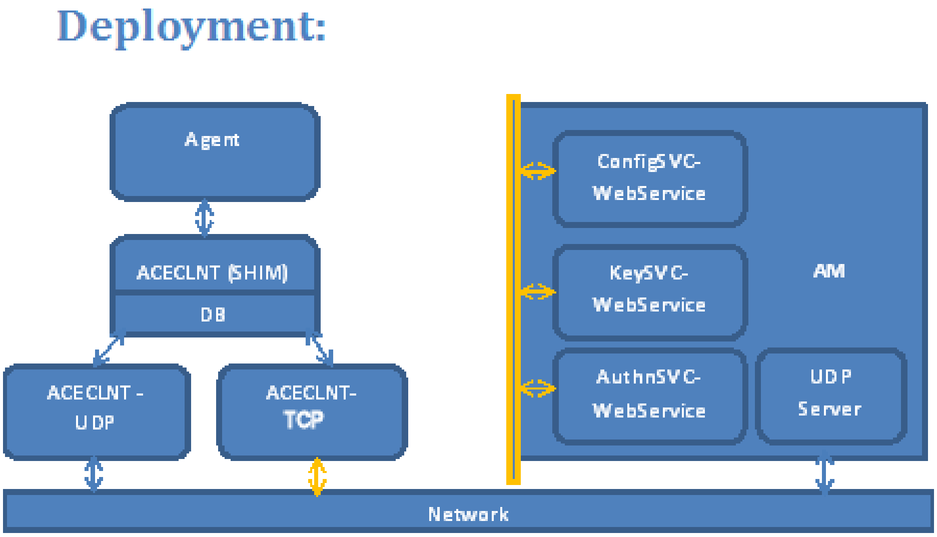
All of the API functions are synchronous, except for the following:

- The Authentication functions AceLock, AceCheck, AceClientCheck, and AceSendNextPasscode
- The PIN Processing functions AceSendPin and AceCancelPin
- The Housekeeping functions AceInit and AceClose

Agents that provide authentication service to multiple users or clients must use the asynchronous functions to remain unblocked while waiting for a response from the Authentication Manager.

For more information, see Appendix B, "Using Synchronous and Asynchronous API Functions."

API Components



An RSA Authentication API 8.5 agent has the following components:

- aceclnt.dll/libaceclnt.so (Shim Layer)
  Loads the corresponding SDK dll based on the environment variable: USEUDP_ENV_VAR (values true/false). The default is TCP.
- aceclnt_tcp.dll/libaceaceclnt_tcp.so
  New TCP library.

- aceclnt_udp.dll/libaceclnt_udp.so
  Old UDP library (not part of the SDK 8.5 kit).
- libaceclnt.a for non-windows static linking
- Available services

  The following services are available in RSA Authentication Manager 8.0 with the TCP agent protocol:
  - **Configuration Service**. Allows agents to retrieve and verify configuration data.
  - **Message Key Service.** Allows agents to negotiate a key that can be used to encrypt subsequent authentications.
  - **Authentication Service**. Processes authentication requests.

API Functions Changed in RSA Authentication API 8.5

The following use cases are no longer supported:

- Offline authentication
  - All APIs won't perform any operation. These APIs will always return success.
  - SD_InitEx 2nd and 3rd parameters are ignored.
  - ACEInitEx 4th parameter is ignored.
- Auto registration of agents
- Login Password Integration
  All APIs are NO-OP.
- Asynchronous APIs
  In SDK 8.5, these are synchronous APIs with call back.

API Functions Not Supported in RSA Authentication API 8.5

The following functions are NO-OP in SDK 8.5 and will always return success:

- AceRefreshIP
- AceDisableNodeSecretCache
- AceGetDAAuthData
- AceGetDAuthenticationStatus
- AceSetLoginPW
- AceGetLoginPW
- SD_Lock
- AceLock

Categories of API Functions

The following table lists the five categories of API functions. .

Category	Purpose	Relevant Functions
Authentication	These functions are directly involved in authenticating a user's identity, mostly by setting values in data structures that are sent to Authentication Manager for validation.	AceCheck AceGetAuthenticationStatus AceSetUsername AceSetPasscode AceClientCheck AceSetUserClientAddress AceContinueAuth AceStartAuth AceSendNextPasscode AceSetNextPasscode AceSetCredential AceSetNextCredential SD_Check SD_CheckCredential SD_ClientCheck SD_ClientCheckCredential SD_Next SD_NextCredential
PIN processing functions	These functions return characteristics of valid PINs, and process PINs using Authentication Manager.	AceCancelPin AceSendPin AceSetPin AceSetPinCredential AceGetAlphanumeric AceGetMaxPinLen AceGetMinPinLen AceGetPinParams AceGetSystemPin AceGetUserSelectable SD_Pin SD_PinCredential
Third-party functions	These functions allow data to be associated with an authentication request but not be processed by Authentication Manager.	AceGetUserData AceSetUserData
Housekeeping functions	These functions initialize, and later clean up the system resources that the API needs.	AceInitialize AceInitializeEx AceInitEx AceShutdown

Category	Purpose	Relevant Functions
		AceCleanup AceClose AceCloseAuth AceInit AceSetTimeout SD_Close SD_Init SD_InitEx
Miscellaneous	Miscellaneous	AceGetShell AceGetTime AceAgentStatusDisplay

Generic Credential API Functions

The following table lists the categories of functions that make up the Generic Credential API. For a description of the Generic Credential API, see “Support for EAP 32 and the Generic Credential API” on page 13.

Category	Purpose	Relevant Functions
Asynchronous authentication	Asynchronous methods include: • Set methods, which immediately update parameters to the specified values. • Action methods, which take a callback function that is used to notify the caller of the current status.	AceSetCredential AceSetPinCredential AceSetNextCredential AceSetClientAddr AceCheck AceClientCheck AceSendPin AceSendNextPasscode
Synchronous authentication	Synchronous methods take all necessary parameters in a single call, send the data to Authentication Manager, and wait for a response.	SD_CheckCredential SD_ClientCheckCredential SD_PinCredential SD_NextCredential

Category	Purpose	Relevant Functions
Authentication attributes	Authentication attributes allow you to request additional information from Authentication Manager. These functions follow the request/response model. With the exception of attributes associated with the New PIN mode or the Next Tokencode mode, these functions are not available until after a successful authentication.	AceSetAuthAttr AceGetAuthAttr
System policy	System policy methods retrieve values set on Authentication Manager that must be used on the client side for EAP 32 and generic credential authentication.	AceGetPepperPolicy AceGetIterCountPolicy AceGetRealmID

C and Java API Equivalents

The following table lists the Java methods from the RSA Authentication Agent API for Java, which correspond to each C function from the RSA Authentication Agent API for C.

Authentication Agent API C Functions	Equivalent Authentication Agent API Java Methods
AceGetAuthenticationStatus	AuthSession.getAuthStatus
AceGetPinParams	AuthSession.getPinData
AceGetShell	AuthSession.getShell
AceGetTime	AuthSession.getAceTime
AceInitialize	AuthSessionFactory.getInstance
AceInitializeEx	AuthSessionFactory.getInstance(path)
AceShutdown	AuthSessionFactory.shutdown
SD_Check	AuthSession.check
SD_ClientCheck	AuthSession.clientCheck

Authentication Agent API C Functions	Equivalent Authentication Agent API Java Methods
SD_Close	AuthSession.close
SD_Init	AuthSessionFactory.createUserSession
SD_Lock	AuthSession.lock
SD_Next	AuthSession.next
SD_PIN	AuthSession.pin
AceGetAlphanumeric	AuthSession.getPinData.isAlphanumeric
AceGetMaxPinLen	AuthSession.getPinData.getMaxPinLength
AceGetMinPinLen	AuthSession.getPinData.getMinPinLength
AceGetSystemPin	AuthSession.getSystemPin
AceGetUserSelectable	AuthSession.getPinData.isUserSelectable
AceAgentStatusDisplay	AuthSessionFactory.getAceServerStatus

Differences between C and Java APIs

The following table lists the differences between C and Java APIs.

Features	C APIs	Java APIs
Synchronous APIs	Yes	Yes
Asynchronous APIs	Yes	No
EAP 32 and the Generic Credential API	Supported	Not supported
Supplies user prompts	Yes	No
Load Balancing	Yes	Yes
Expiration Facility and Cleanup Callbacks	Yes	No
Thread Safety	Yes	Yes
AceInitialize Error Detection	Yes	No
Data Encapsulation	Yes	Yes

2 Using the RSA Authentication Agent API

The functions described in this guide are demonstrated in sample programs available at RSA SecurCare Online **https://knowledge.rsasecurity.com**. RSA recommends that you examine the source code for each of the samples to understand the proper calling sequences. Each sample demonstrates a different set of functions.

System Requirements

The following tables list supported platforms, versions, and binary compatibility for this release.

Platform Support

Microsoft Windows

Version	Binary Compatibility
Windows XP Windows 2003 Enterprise SP2 Windows 2008 Enterprise SP2 Windows Vista Windows 7 SP1 Windows 8	32-bit and 64-bit
Windows 2003 R2 SP2 Windows 2008 R2 Enterprise SP2 Windows 2012 Server	64-bit

UNIX

Version	Binary Compatibility
AIX 6.1 (PPC) AIX 7.1 (PPC) HPUX 11i v3 (Itanium) Sun Solaris 10 (SPARC) Sun Solaris 10 (Intel) Sun Solaris 11 (SPARC) Sun Solaris 11 (Intel) SuSE Enterprise Linux 10 SuSE Enterprise Linux 11 Red Hat Enterprise Linux 4.8 Red Hat Enterprise Linux 5.9 Red Hat Enterprise Linux 6.3	32-bit and 64-bit

The Authentication API requires the C runtime library **msvcr80.dll** which gets installed as part of the Visual Studio 2005 Redistributable Package (x86 | x64).

Authentication Agent API is supported on VMware ESX 4 and ESX 5 with the following operating systems.

Hosted Operating System	VMware ESX 4.x	VMware ESX 5.x
Windows XP (32-bit & 64-bit)	✕	✕
Windows 2003 Enterprise SP2 (32-bit & 64-bit)	✕	✕
Windows 2003 R2 Enterprise SP2 (32-bit & 64-bit)	✕	✕
Windows 2008 Enterprise SP2 (32-bit & 64-bit)	✕	✕
Windows Vista (32-bit & 64-bit)	✕	✕
Windows 7 SP1 (32-bit & 64-bit)	✕	✕
Windows 2008 R2 Enterprise SP2 (64-bit)	✕	✕
Windows 8 (32bit & 64-bit)	✕	✕
Windows 2012 Server (64-bit)	✕	✕
Red Hat Enterprise Linux 4.8 (32-bit & 64-bit)	✕	✕
Red Hat Enterprise Linux 5.9 (32bit & 64 bit)	✕	✕
Red Hat Enterprise Linux 6.3 (32-bit & 64-bit)	✕	✕

Hosted Operating System	VMware ESX 4.x	VMware ESX 5.x
Solaris 10 (64-bit)	×	×
Solaris 11 (64-bit)	×	×
SuSE Enterprise Linux 10 (32-bit & 64-bit)		×
SUSE Enterprise Linux 11 (32-bit and 64-bit)	×	×

Compatibility

The 8.5 APIs and SDK are not compatible with the UDP protocol (and therefore, cannot communicate with Authentication Manager 6.1 and 7.1, because they only support the UDP protocol). Authentication Manager 8.x, however, supports both TCP and UDP protocols, as shown in the following table.

RSA Authentication Manager Version	8.1 APIs	8.5 APIs
6.1	UDP	
7.1	UDP	
8.0	UDP	TCP
8.x	UDP	TCP

Backward Compatibility

The 8.5 SDK is compatible with existing agent integration code, in most cases. There are, however, some function calls from previous SDK versions whose function has been changed or eliminated from the 8.5 SDK. For a list of calls and which versions support them, see Appendix A, “Return Values and Result Codes”.

RSA ships the following two libraries with SDK 8.5:

- **Shim library**. (aceclnt.dll/libaceclnt.so) A shim library, used outside of the agent code, exposes the same interfaces as SDK 8.5 and determines whether the SDK 8.5 library, (aceclnt_tcp.dll/libaceclnt_tcp.so) is used or the aceclnt_udp.dll/libaceclnt_udp.so library is used based on the environment variable, USEUDP_ENV_VAR. If the environment variable is set to TRUE, aceclnt_udp.dll is used; FALSE or not set, aceclnt_tcp.dll is used.

Note: aceclnt_udp.dll/libaceclnt_udp.so is not part of this kit. To use old SDK with shim layer, get the aceclnt.dll/libaceclnt.so from an older kit and rename to aceclnt_upd.dll/libaceclnt_udp.so.

- **SDK 8.5 library**. (aceclnt_tcp.dll/libaceclnt_tcp.so) The SDK 8.5 library can be used independently of the shim library. To use aceclnt_tcp.dll/libaceclnt_tcp.so without the shim layer, back up the shim library (aceclnt.dll/libaceclnt.so). Rename aceclnt_tcp.dll/libaceclnt_tcp.so to aceclnt.dll/libaceclnt.so.

  Note: You must use the shim layer in order to switch between aceclnt_udp.dll and aceclnt_tcp.dll using the env variable, USEUDP_ENV_VAR.

RSA Authentication SDK 8.5 provides the following options for backward compatibility with older versions of the APIs:

- Integrate with SDK 8.1, SP1 Only
- Integrate Libraries from SDK 8.5 and SDK 8.1
- Integrate with SDK 8.5 only

Integrate with SDK 8.1, SP1 Only

Integrate with SDK 8.1 without using the new features of SDK8.5. No changes are required to your agent code. Existing integrations will continue to work in the same way as before.

Note: The RSA Ready Partners certification program cannot accept applications if the corresponding code fails to support two-factor authentication.

Integrate Libraries from SDK 8.5 and SDK 8.1

To integrate libraries from both SDK 8.1 and SDK 8.5:

1. Rename the existing shipping RSA library to **aceclnt_udp.dll (**for Linux, libaceclnt_udp.so).
2. Use the **aceclnt.dll** (shim) (for Linux, libaceclnt.so) available with SDK 8.5 and **aceclnt_tcp.dll** (for Linux, libaceclnt_tcp.so) in the distribution.
3. Set the environment variable, USEUDP_ENV_VAR. The shim library uses the environment variable to determine which library to use. The following settings apply:
   - TRUE = aceclnt_udp.dll
   - FALSE= aceclnt_tcp.dll
   - Not Set (default)= aceclnt_tcp.dll

Note: aceclnt_udp.dll is not part of the new SDK 8.5 kit; it is available with 8.1 installations.

Integrate with SDK 8.5 only

To integrate with SDK 8.5 only, use the new tcp protocol, **aceclnt_tcp.dll** without the shim layer. You will only be able to integrate libraries with Authentication Manager 8.0. You will not be able to switch between aceclnt_tcp.dll and aceclnt_udp.dll using the environment variable, USEUDP_ENV_VAR.

To use aceclnt_tcp.dll without the shim layer:

- Back up the shim library, aceclnt.dll and rename it.
- Rename aceclnt_tcp.dll to aceclnt.dll.

Static Linking

If the code is statically linked to RSA libraries (non-Windows platforms) you cannot switch between SDK 8.1 and SDK 8.5 using the environment variable.

To use the new library, rebuild the code by replacing the old libaceclnt.a with the new libaceclnt.a. Use the g++ compiler.

Note: To use the gcc compiler, link with the libstdc++ library and use the following flag while linking, -lstdc++

Working with the APIs

Software Development Kit Contents

The **Auth SDK** folder contains Release Notes and the following folders:

- **doc.** Contains the Developer's Guide.
- **inc.** Contains all the header files.
- **lib.** Contains two subfolders: 32-bit and 64-bit. These folders contain the libraries required for all supported platforms.

Note: For the Windows platform, there are two versions of the authapi binaries: **<kit_root>\lib\[32bit|64bit]\nt\Release**

This version of the **aceclnt.dll** is built with the /MD compiler option and uses the dll version of the C Runtime library. Hence it requires the Visual Studio 2005 C Runtime package to be installed on the target machine.
<kit_root>\lib\[32bit|64bit]\nt\Release_MT

This version of the **aceclnt.dll** is built with the /MT option compiler option and is statically linked to the Visual Studio 2005 C Runtime library.

- **samples.** Contains the sample code, sync/async samples +acestatus utility. Contains four samples that illustrate the application of the API in all modes of operation. You can use these samples to test and run the function calls. Also contains sample code **acestatus**, which can be used to check the status of the Authentication Manager. For more information, see "Running the Sample Code" on page 36.
- **src.** Contains sdmsg.mc and sdmsg.msg. For more information, see Appendix C, "Modifying the Message Catalog" on page 203.
- **util.** The acestatus utility is in the samples folder.

Building the Agent

Before You Begin

See the section "Deployment Guidelines" on page 205, for deployment-specific guidelines and important information.

To build the agent:

1. Extract the C Authentication SDK files to the required location.
2. Build your custom agent.

Setting Up the Agent

To set up the agent:

1. From the **lib folder**, access to the required platform-specific folder. On Windows machines, copy the libraries to the **system32** folder. On UNIX machines, ensure that the environment variable **LD_LIBRARY_PATH** points to the location where the libraries are kept.
2. Register the agent as an agent host in the Authentication Manager server. For information about deploying IPv4/IPv6 agents, see the *RSA Authentication Manager 8.x Administrator's Guide.*
3. Copy the **sdconf.rec** file to the required location. Use the **rsa_api.properties** file to specify the location of the **sdconf.rec** file.
   - For Windows: the **rsa_api.properties** file is in the same location as aceclnt.dll.
   - For Linux: the **rsa_api.properties** file is in the **/var/ace** folder (default location). You can also set the environment variable, VAR_ACE to the directory where the **rsa.api.properties** file is located.

   Note: For Linux, the VAR_ACE environment variable can be used to specify the location of the **sdconf.rec** file if you do not want to use the **rsa_api.properties** file.
4. If you are using manual load balancing, copy the **sdopts.rec** file to the same location as the **sdconf.rec** file.
5. Run the agent.

Upgrading from 8.1

To upgrade an agent created with 8.1 APIs to the 8.5 APIs, you must complete the following tasks.

- Upgrade the APIs to 8.5.
- Renew the node secret (required only if you are upgrading from RSA Authentication Agent API 8.1 or earlier).
- Verify authentication with the Authentication Manager.

Upgrade the APIs to SDK 8.5

To upgrade the APIs to 8.5:

1. Identify the location of the existing C SDK libraries.
2. Extract the 8.5 C Authentication SDK files to a different local location.
3. From the **lib** folder, drill down and copy the required platform-specific folder.
4. Replace the existing platform-specific folder with the new folder from the 8.5 C Authentication SDK.
5. Relink with the current API libraries. For information, see "Compiling and Linking with the API" on page 37.

Note: If there are multiple agents on the same host, upgrade options depend on the location of binaries and configuration files. You must ensure that the upgrade of one agent does not break the other agent. For more information, see "Multiple Agents on a Host Authenticating with a Common Authentication Manager" on page 205.

Renew the Node Secret

Note: If you are upgrading from RSA Authentication Agent API 8.1 SP1, you do not need to renew the node secret. These steps are required only if you are upgrading from RSA Authentication Agent API 8.1 or earlier.

Authentication Manager uses the node secret to verify the identity of IPv4/IPv6 authentication agents. IPv4/IPv6 authentication agents do not require a node secret.

To renew the node secret:

1. Delete the node secret file (securid file) from the agent.
2. Clear the node secret from the Authentication Manager server.
3. Load a new node secret.

Load a New Node Secret

You can manually create a node secret in the Authentication Manager server and export it to the agent host. Once the node secret file is imported into the agent host machine, you must run the **agent_nsload** utility to extract the node secret file and store it appropriately. The node secret can be stored either in the default path or in a user-defined path.

Note: The **sdconf.rec** file must be present in the destination folder on the host machine before you can extract and load the node secret file.

Note: On Windows, you must be an administrator to run the **agent_nsload** utility.

Extract the Node Secret to the Default Location

1. To run the a**gent_nsload** utility to extract the node secret to the default location, type:
   - On UNIX: agent_nsload -f /default_dir/nodesecret.rec
   - On Windows: agent_nsload -f C:\default_path\nodesecret.rec

   You will be prompted for the password.
2. Type the password and press **Enter**.

Extract the Node Secret to a User-Defined Location

1. To run the **agent_nsload** utility to extract the node secret to a user-defined location, type:
   - On UNIX: agent_nsload -f /VAR_ACE/nodesecret.rec -d VAR_ACE/new_dir/
   - On Windows: agent_nsload -f C:<windows path>\System32\nodesecret.rec -d C:\<windows path>\System32\new_dir\

   You will be prompted for a password.
2. Type the password and press **Enter**.

Critical and Sensitive Files

In addition to the Authentication API binaries (aceclnt.[so|dll], aceclnt_tcp.[so|dll], sdmsg.[dll|so|mc|cat]), the Authentication API maintains critical and sensitive files, as listed below. The application using the SDK must restrict access to these files. One of the ways to restrict access is setting the appropriate file permissions on the files and the complete directory structure in which they reside, before invoking the API.

sdconf.rec. This file is generated by the Authentication Manager server, and contains configuration information that controls the behavior of the Authentication API. Non-privileged users must not be able to read or edit this file. This file is a read-only file. Agent never tries to edit this file.

securid. This file contains a shared secret key used for agent authentication. Non-privileged users must not be able to read or edit this file. This is a read-only file. Agent never tries to edit this file.

config.xml, bootstrap.xml and root.cer. These files contain the configuration information and root certificate that agent receives from AM. These files should not be edited manually. Only the agent process should create/override these files.

Agent API Configuration

You must register the agent as an agent host in the Authentication Manager server. For information about deploying IPv4/IPv6 authentication agents, see the *RSA Authentication Manager 8.x Administrator's Guide.*

sdconf.rec

You can use the **sdconf.rec** file for both the UDP and TCP protocols. The **sdconf.rec** file should not be edited.

rsa_api.properties Example

Configuring the **rsa_api.properties** file is optional.

Location of **rsa_api.properties** file:

- On Windows: the **/samples** folder of the kit.
- On Linux: the **/var/ace** folder or the location specified by the environment variable, VAR_ACE.

Key	Description	Acceptable values
SDCONF_LOC	Indicates the path to the Authentication Manager configuration file, **sdconf.rec.** Changes made to this file take effect only on initialization/restart	The applicable path. For example: **/var/ace/api/sdconf.rec or C:\WINDOWS\system32\sdconf.rec** Default path of sdconf.rec if not specified in **rsa_api.properties** Windows: **C:\Windows\System32** Non windows: The location specified by VAR_ACE environment variable. If VAR_ACE is also not set then default location is **/var/ace/**.

Key	Description	Acceptable values
SDOPTS_LOC	Indicates path to configuration file **(sdopts.rec)** used to configure load balancing. For more information, see "Configuring the sdopts.rec File" on page 39. **Note:** Default dynamic load balancing works even when sdopts.rec is not used. Changes made to this file take effect only on initialization/restart.	The applicable path. For example: **/var/ace/api/sdopts.rec or C:\WINDOWS\system32\sdopts.rec**
RSA_CONNECTION_TIMEOUT	Connection timeout for server connection in seconds.	The default value will be taken from **config.xml**.
RSA_READ_TIMEOUT	Read timeout for server connection in seconds.	The default value will be taken from **config.xml**.
RSA_AGENT_NAME	Indicates name of agent as configured in AM.	This should be set to a valid agent-name configured in AM. Default value is the hostname of agent machine.
RSA_AGENT_TYPE	Indicates the type of Agent which is using this SDK.	RSA_AGENT_TYPE = RSA_Web_Agent Default value will be 'UnKnown'.
RSA_AGENT_VERSION	Indicates the version of Agent which is using this SDK.	RSA_AGENT_VERSION=7.1 Default value will be 'UnKnown'.
RSA_AGENT_PLATFORM	Indicates the platform in which the Agent is running.	RSA_AGENT_PLATFORM = Windows_Server_2012 Default value will be 'UnKnown'.

Key	Description	Acceptable values
SDNDSCRT_LOC	Path to node-secret file. Node-secret should be downloaded from AM separately and agent_nsload utility should be used to decrypt the downloaded file. SDNDSCRT_LOC should point to decrypted “securid” file. For using client authentication using node-secret the feature should be enabled for particular agent in AM. **Note:** If you do not use the rsa_api property file, you cannot use a node secret generated using SDK 8.1 SP2.	The applicable path. For example: **/var/adm/ace/api/securid or C:\WINDOWS\system32\securid**
RSA_CONFIG_DATA_LOC	Folder name where configuration files will be stored. These configuration files should not be edited. The config files stored in this folder are: • **config.xml** • **bootstrap.xml** • **root.cer** If the tag is not specified, then the config files will be created in a folder having name as agentname. The folder location will be location specified by VAR_ACE env varilable in case of non-windows and location of aceclnt.dll in case of windows	A valid path where config will be stored. For example: **RSA_CONFIG_DATA_LOC = C:\RSA\**

Key	Description	Acceptable values
RSA_ENC_ALGLIST	Used to specify the list of encryption algorithms to be used for encryption while communicating with AM. If this is not specified then automatically the best algorithm supported by AM and agent is chosen.	This is the default list of algorithms: RSA_ENC_ALGLIST=AES /24,AES/32,AES/16 This list is used for algorithm negotiation when nothing is specified. You can delete algorithms in this list to make sure only one of the remaining algorithms is used. **Note:** No additional algorithm can be added to this list.
RSA_LOG_FILE_LOC	Folder name where the log files should be generated or complete filename with path of log file.	All valid directory or file location.
RSA_LOG_LEVEL	Log level.	Set your log level to either of these values "info","warn","error", "verbose". The default log level if the tag is not set is "warn".
RSA_LOG_FILE_SIZE	Log file size.	The maximum size is 1MB. If this tag is not provided the default size used will be 1 MB.
RSA_LOG_FILE_COUNT	Number of log files to be created before log file rotation.	If this tag is not provided a default value of 10 will be used.

Perform a Test Authentication

You can use the code available in the sample code files to perform an authentication. See "Running the Sample Code" on page 36 for more information.

Running the Sample Code

The RSA Authentication Agent API includes the sample source code, **<AuthSDK-kit>\samples** that demonstrates the basic calling sequence and use of synchronous API function calls.

The samples include:

\<AuthSDK-kit>\samples\async. Demonstrates the asynchronous API calls. The sample illustrates the calling mechanisms required when using the asynchronous API. It does not illustrate this in the context of an asynchronous application.

\<AuthSDK-kit>\samples\sync. Demonstrates use of the synchronous API calls AceStartAuth, AceContinueAuth, and AceCloseAuth, which simplify the processing of RSA SecurID authentication.

\<AuthSDK-kit>\samples\sync2. Demonstrates use of the thread-safe synchronous API calls SD_Close, SD_Init, SD_Next, and SD_Pin, which you can use to port existing products with built-in RSA SecurID support to the new API library.

\<AuthSDK-kit>\samples\sync3. Demonstrates how to upgrade code that uses 6.0 and earlier calls. Uses the same synchronous API calls that are used in the **sync2** example.

\<AuthSDK-kit>\samples\acestatus. You can verify the status of the Authentication Manager by running the code given in the sample code file **acestatus**. This returns the status of each Authentication Manager on which the agent is registered as an agent host, and details including server name.

The following table lists the information displayed in the Authentication Manager section.

Returned Information	Description
Configuration Version	The version of the **sdconf.rec** file that is in use.
DES Enabled	If your configuration environment supports legacy protocols, YES is displayed.
Client Retries	The number of times the agent sends authentication data to Authentication Manager before a time-out occurs.
Client Timeout	The amount of time (in seconds) that the agent waits before res ending authentication data to Authentication Manager.
Server Release	The version number of Authentication Manager.
Communication	The protocol version used by Authentication Manager and the agent.

The following table lists the status information displayed in the Authentication Manager section.

Status Information	Description
Server Active Address	Server name that the agent uses to communicate with the server

The following table lists the server status information displayed in the Authentication Manager section.

Server Status	Description
Available for Authentications	This server is available to handle authentication requests.
Unused	The server has not yet received an authentication request.
For Fail over only	The server is reserved for fail over use only.
Default Server During initial requests	Only this server is available to handle requests at this time.

Note: RSA recommends that you download the most recent updates to the sample source code files from RSA SecurCare Online at **https://knowledge.rsasecurity.com**.

To run the sample code, execute the required binary by drilling down to the required sample and platform.

For more information about running the statically and dynamically linked samples, see "Using Dynamic Libraries or Static Libraries in UNIX" on page 38.

Compiling and Linking with the API

To compile and link the APIs, perform the following tasks depending on the operating system used.

Compiling on UNIX

To compile an agent using the Authentication API:

1. Add the **acexport.h** file to your source.
2. Define the symbol **UNIX** with the **-DUNIX** compiler option.

To link objects with the API library:

From **<AuthSDK-kit>/lib**, select the correct platform of the API library: link to **libstdc++.**

Compiling on Windows

To compile an agent using the Authentication API:

- Include **acexport.h** in your source.
- Define the symbol **WIN32** with the **-DWIN32** compiler option.

Using Dynamic Libraries or Static Libraries in UNIX

The API library provides both dynamic and static (archive) libraries for all supported UNIX platforms. When using dynamic libraries to build agent applications in the case of AIX, make sure to use the correct version of the **libaceclnt.a** file with which you compiled the code.

To make sure that you use the correct version of libaceclnt.so:

Enter the full path to the file in the command line passed to the **ld** command. If the version of the dynamic linker that you use provides options to disable the library path processing at runtime, you must use those options as well.

For examples, see the makefiles provided in **\<AuthSDK-kit>\samples**.

For information about downloading the latest sample files from the RSA SecurCare Online, see "Running the Sample Code" on page 36.

To execute 64-bit samples:

1. In the makefile, update LIBDIR and LIBPATH to point to the location where you placed the files **libaceclnt.so**, **libaceclnt_tcp.so**, **libxerces-c-3.1**, and **sdmsg.cat**.
2. For all non-windows platforms except AIX:
   Set LD_LIBRARY_PATH=<path to 64 bit aceclnt libs

   For AIX:
   Set LIBPATH=<path to 64 bit aceclnt libs>
3. For AIX rename **libaceclnt.so** to **libaceclnt.a**.
4. Change the directory to the 64-bit samples directory.
5. Run the dynamically linked sample, for example, **./async_so**.

To execute 32-bit samples:

1. In the makefile, update LIBDIR and LIBPATH to point to the location where you placed the files **libaceclnt.so**, **libaceclnt_tcp.so**, **libxerces-c-3.1**, and **sdmsg.cat**.
2. For all non-windows platforms except AIX:
   Set LD_LIBRARY_PATH=<path to 32 bit aceclnt libs

   For AIX:
   Set LIBPATH=<path to 32bit aceclnt libs>
3. For AIX rename **libaceclnt.so** to **libaceclnt.a**.
4. Change the directory to the 32-bit samples directory.
5. Run the dynamically linked sample, for example, **./async_so**.

Using the sdopts.rec File

This section describes the components that you can use to create an **sdopts.rec** file.

Creating the sdopts.rec File

To create the sdopts.rec file:

1. Using a text editor, create an **sdopts.rec** file and store it in the default application directory where you stored the **sdconf.rec** file.
2. Configure the **sdopts.rec** file to be read-only.

Note: On UNIX platforms, you must set the required file permission.

3. Specify the location of the **sdopts.rec** file in the rsa_api.properties file.

Configuring the sdopts.rec File

Note: Each time you modify the **sdopts.rec** file, you must restart the agent to register your changes.

The **sdopts.rec** file can contain the following types of lines:

- Comment lines, each beginning with #
- Keyword-value pairs, which can be any of the following:
  - AVOID = <*DNS-resolvable server name*> or <*IPv6 or IPv4 address*>

    Specifies an Authentication Manager server to exclude for dynamic or round-robin load balancing. <*DNS-resolvable server name*> or <*IPv6 or IPv4 address*> is one of the Authentication Manager servers configured as primary or replica. To exclude multiple servers, use multiple AVOID keywords in the **sdopts.rec** file.

    Note: The AVOID and USESERVER keywords are mutually exclusive and cannot be used together with the same Authentication Manager server in **sdopts.rec**. Both AVOID and USESERVER keywords support IPv6 and IPv4 address support.

  - USESERVER = <*DNS-resolvable server name*> or <*IPv6 or IPv4 address*>, *priority*

    Specifies an Authentication Manager server to receive authentication requests from the authentication agent host according to a specified *priority* value.

    Including this value in the **sdopts.rec** file enables manual load balancing.

    Use one USESERVER setting for each Authentication Manager server that the authentication agent host uses. The combined maximum number of Authentication Manager servers you can specify in the **sdopts.rec** and **sdconf.rec** files is 11.

Note: The new TCP protocol supports the USESERVER keyword only. Tags such as CLIENT_IP, ALIAS are no longer supported.

Each USESERVER keyword-value must consist of the server name of the Authentication Manager server separated by a comma from the assigned priority. The priority specifies if or how often an Authentication Manager server receives authentication requests. You can list the IP addresses or server names in the **sdopts.rec** file in any order, but you must list each separately, one per line.

The following table lists the priority values that you can specify.

Priority	Description
2-10	Send authentication requests to the Authentication Manager server using a randomized selection based on the assigned priority of the Authentication Manager server. The range is from 2 to 10. The higher the value, the more requests the Authentication Manager server receives. For example, a priority of 10 will receive many more requests than a priority of 2.
1 or 0	Ignore this Authentication Manager server. **Note:** A priority of 1 or 0 means the Authentication Manager server will not be used to authenticate the agent unless all Authentication Manager servers with priorities of 2 to 10 listed in the **sdopts.rec** file are known to the agent as unusable. By specifying a priority value of 1 or 0, you can include an entry in **sdopts.rec** for an Authentication Manager server without using it. If you later decide to use the server, you can change the priority value.

You must assign a priority to each Authentication Manager server that you add to the **sdopts.rec** file. Otherwise, the entry is invalid.

Logging

To enable logging, some configuration properties must be set in the **rsa_api.properties** file. The configuration parameters are RSA_LOG_FILE_LOC, RSA_LOG_LEVEL, RSA_LOG_FILE_SIZE, and RSA_LOG_FILE_COUNT. For descriptions of these tags, see "rsa_api.properties Example" on page 32.

3 API Functions

This chapter describes the RSA Authentication Agent API functions in detail.

To understand the proper calling sequences required, RSA recommends that you examine the source code for each of the samples. For more information, see "Running the Sample Code" on page 36.

Note: The entry point prototypes contain the WINAPI calling convention of Windows. However, you can compile these prototypes for both Windows and UNIX. For additional information, see "Compiling and Linking with the API" on page 37.

AceAgentStatusDisplay

Description

int WINAPI AceAgentStatusDisplay(S_status_display * dsp)

The S_Status_display structure is defined in **status_display.h**. The AceAgentStatusDisplay function obtains the last known status and configuration information for the realm. This function assists in writing customized utility applications, to check the configuration and connection status, and to perform test authentication.

Parameters

	Parameter	Description
[out]	dsp	The structure is populated with data.

Return Values

Returned Value	Description
ACE_SUCCESS	The call was successful.
ACE_INVALID_ARG	An invalid argument was supplied to an API function.

Requirements

Minimum supported Authentication Manager Server Version	6.1 or later
Category	Synchronous API
Header	status_display.h
Static Library	UNIX: libaceclnt.a
Shared Library	Windows: • aceclnt.dll • aceclnt_tcp.dll • sdmsg.dll • xerces-c_3_1_vc80.dll UNIX: • libaceclnt.so • libaceclnt_tcp.so • libxerces-c-3.1.so AIX: • libaceclnt.so **Note:** Rename to libaceclnt.a. For more information, see "Using Dynamic Libraries or Static Libraries in UNIX" on page 38. • libaceclnt_tcp.so • libxerces-c-3.1.a

If the value returned is ACE_SUCCESS, dsp populates with the following data in its structure:

u32Size: Stores the size of this structure

config_version: Version number of the RSA Authentication Manager

acmmaxservers: Maximum number of servers from SDCONF.REC

acmmaxreplicas: Maximum number of replica servers may be as high as MAXCONFIG_SERVERS

acmmaxretries: The maximum number of retries allowed for agent communication with the RSA Authentication Manager.

acmbasetimeout: The number of seconds that agent waits for a response from the RSA Authentication Manager before timing out.

use_des: Use DES or SDI algorithm for encryption.

trusted: Identifies whether name resolution is to be trusted.

acmport: The UDP port of the RSA Authentication Manager authentication service.

acmservice[32]: The service name of the RSA Authentication Manager authentication service.

acmprotocol[4]: The protocol used to resolve the RSA Authentication Manager authentication

service port.
server_hi_protocol: Highest message protocol version RSA Authentication Manager understands
default_alias_options: The default values for alias options in either the sdconf.rec or the sdopts.rec, depending on which file is more recent.
server_release_from_server[4]: The RSA Authentication Manager release number. It is a byte array containing the major, minor, patch, and build versions of the RSA Authentication Manager.

Note: If the first byte of server_release_from_server is 0, the agent has not yet received the release number from the Authentication Manager. Two successful authentications are required before the Authentication Manager can return its version.

DISP_SRVR_INFO acm_servers[DISP_MAX_SERVERS]: Contains an object array containing the status of all Replicas in the realm.

AceCancelPin

Description

```
int WINAPI AceCancelPin(
  SDI_HANDLE SdiHandle,
  void (WINAPI*appCallback)(SDI_HANDLE SdiHandle));
```

The AceCancelPin function cancels the pending new PIN operation associated with SdiHandle. AceCancelPin is called only when the result code from AceCheck is ACM_NEW_PIN_REQUIRED and the user has chosen not to proceed through New PIN mode.

Architecture

This asynchronous function returns immediately with a status value. If the status is ACE_PROCESSING, threads are enabled to process the data.

The AceCancelPin function attempts to find the data associated with this specific authentication process through the unique handle value. If the data is found, AceCancelPin calls another thread to process the data.

When the threads invoked by AceCancelPin finish, the developer-defined function is called (if one is or was supplied). AceCancelPin allows the caller to defer changing the PIN in the Authentication Manager token record.

Parameters

	Parameter	Description
[in]	SdiHandle	The value of a handle originally assigned by a call to AceInit.
[in]	appCallback	A pointer to a function written by the developer. This function does not return any value. The handle to the data specific to the authentication process is passed to this function. A NULL value can be passed instead of a valid function pointer.

Return Values

Returned Value	Description
ACE_PROCESSING	Normal return.
ACE_ERR_INVALID_HANDLE	AceCancelPin could not locate the data associated with the handle.
ACE_TOO_MANY_CALLERS	An attempt was made to use a handle while a call was already in progress. This happens if a multithreaded application uses the same handle from multiple threads at the same time. This return code is not valid in this version.

Error Handling

To handle errors appropriately, use the value returned by this function at a decision point in your code. A successful return value allows the processing to continue.

Checking the Results

A caller can verify the result of the authentication operation by calling the AceGetAuthenticationStatus function once the requested operation has been completed.

ACM_OK	The PIN operation was successfully aborted.

Requirements

Minimum supported Authentication Manager Server Version	6.1 or later
Category	Asynchronous API
Header	acexport.h
Static Library	UNIX: libaceclnt.a
Shared Library	Windows: • aceclnt.dll • aceclnt_tcp.dll • sdmsg.dll • xerces-c_3_1_vc80.dll UNIX: • libaceclnt.so • libaceclnt_tcp.so • libxerces-c-3.1.so AIX: • libaceclnt.so **Note:** Rename to libaceclnt.a. For more information, see "Using Dynamic Libraries or Static Libraries in UNIX" on page 38. • libaceclnt_tcp.so • libxerces-c-3.1.a

AceCheck

Description

```
int WINAPI AceCheck(
  SDI_HANDLE SdiHandle,
  void (WINAPI*appCallback)(SDI_HANDLE));
```

The AceCheck function checks the validity of the credential (previously set by a call to AceSetPasscode or AceSetCredential) for the user name (previously set by a call to AceSetUsername). AceCheck is called only after successful calls to AceInit, AceSetPasscode, and AceSetUsername for the specific authentication request.

Architecture

This asynchronous function returns immediately with a status value. If the status is ACE_PROCESSING, other threads perform the rest of the task.

The AceCheck function attempts to find the data associated with this specific authentication process through the unique handle value. If the data is found, AceCheck enables other threads to process the data. When the threads invoked by AceCheck finish, the developer-defined function is called (if one has been supplied).

Parameters

	Parameter	Description
[in]	SdiHandle	The value of a handle originally assigned by a call to AceInit.
[in]	appCallback	A pointer to a function written by the developer. This function returns nothing. The handle to the data area specific to the authentication process is passed to this function. A NULL value can be passed instead of a valid function pointer.

Important: There is a design constraint on the use of SDI_HANDLE. An asynchronous application of the API must avoid concurrent use of the same SDI_HANDLE. Multiple threads using the same SDI_HANDLE return incorrect results.

Return Values

Returned Value	Description
ACE_PROCESSING	Normal return.
ACE_ERR_INVALID_HANDLE	AceCheck could not locate the data associated with the handle.
ACE_UNDEFINED_PASSCODE	The passcode has yet to be set for the authentication request.
ACE_UNDEFINED_USERNAME	The user name has yet to be set for the authentication request.
ACE_TOO_MANY_CALLERS	An attempt was made to use a handle while a call was already in progress. This happens if a multithreaded application uses the same handle from multiple threads at the same time. This return code is not valid in this version.

Error Handling

To handle errors appropriately, use the value returned by this function at a decision point in your code. A successful return value allows the processing to continue.

Checking the Results

A caller can verify the result of the authentication operation by calling the AceGetAuthenticationStatus function once the requested operation is completed.

Result	Description
ACM_OK	The user was successfully authenticated. Use the AceGetShell function to access the shell field.
ACM_ACCESS_DENIED	The user failed authentication.
ACM_NEXT_CODE_REQUIRED	Next tokencode required. Use AceSetNextPasscode and AceSendNextPasscode to complete the transaction. The AceCheck API must be called within 140 seconds if using Authentication Manager 6.1.2, or 120 seconds if using Authentication Manager 7.1. If using Authentication Manager 8.0, the timeout is 180 seconds.

Result	Description
ACM_NEW_PIN_REQUIRED	New PIN required. You can use the AceGetxxx functions to access the PIN limits. Use AceSetPin and AceSendPin, or AceCancelPin to complete the transaction. These APIs must be called within 140 seconds if using Authentication Manager 6.1.2, or 120 seconds if using Authentication Manager 7.1. If using Authentication Manager 8.0, the timeout is 180 seconds.

Requirements

Minimum supported Authentication Manager Server Version	6.1 or later
Category	Asynchronous API
Header	acexport.h
Static Library	UNIX: libaceclnt.a
Shared Library	Windows: • aceclnt.dll • aceclnt_tcp.dll • sdmsg.dll • xerces-c_3_1_vc80.dll UNIX: • libaceclnt.so • libaceclnt_tcp.so • libxerces-c-3.1.so AIX: • libaceclnt.so **Note:** Rename to libaceclnt.a. For more information, see “Using Dynamic Libraries or Static Libraries in UNIX” on page 38. • libaceclnt_tcp.so • libxerces-c-3.1.a

AceCleanup

Description

```
void WINAPI AceCleanup(
  void (WINAPI*appCallback)(SDI_HANDLE));
```

The AceCleanup function can be called at any time to discard every authentication request in progress. After this call is returned, every handle that was returned from a call to AceInit, AceStartAuth, or SD_Init becomes invalid. The callback argument is used to clean up before the authentication handle is discarded.

If the callback passed as an argument to AceCleanup is null, then any callback previously defined by a call to AceSetTimeout is invoked.

Architecture

This synchronous function checks the internal list of outstanding authentication handles and performs the clean-up operation on them. This has the same effect as if the user's actions had caused the appropriate final function (AceClose, AceCloseAuth, or SD_Close) to be called.

Parameters

	Parameter	Description
[in]	CleanupCallback	A pointer to a function written by the developer. This function returns nothing. The handle to the data area specific to the authentication process is passed to this function. A NULL value can be passed instead of a valid function pointer.

Return Values

The function does not return a value. When the function call returns, every outstanding authentication has been discarded, and all the handles that were returned by AceInit, AceStartAuth, or SD_Init are invalid.

Requirements

Minimum supported Authentication Manager Server Version	6.1 or later
Category	Asynchronous API
Header	acexport.h
Static Library	UNIX: libaceclnt.a
Shared Library	Windows: • aceclnt.dll • aceclnt_tcp.dll • sdmsg.dll • xerces-c_3_1_vc80.dll UNIX: • libaceclnt.so • libaceclnt_tcp.so • libxerces-c-3.1.so AIX: • libaceclnt.so **Note:** Rename to libaceclnt.a. For more information, see "Using Dynamic Libraries or Static Libraries in UNIX" on page 38. • libaceclnt_tcp.so • libxerces-c-3.1.a

AceClientCheck

Description

```
int WINAPI AceClientCheck(
  SDI_HANDLE SdiHandle,
  void (WINAPI*appCallback)(SDI_HANDLE));
```

The AceClientCheck function checks the validity of the credential (previously set by a call to AceSetPasscode or AceSetCredential) for the user name (previously set by a call to AceSetUsername) on behalf of another client. The other client's IP address must have been set with a call to AceSetUserClientAddress. AceClientCheck is called only after successful calls to AceInit, AceSetPasscode, AceSetUsername, and AceSetUserClientAddress for the same authentication request.

Architecture

This asynchronous function returns immediately with a status value. If the status is ACE_PROCESSING, other threads perform the rest of the task.

The AceClientCheck function attempts to find the data associated with the specific authentication process through the unique handle value. If the data is found, AceClientCheck checks to make sure that all the data it needs to attempt authentication is present, and then enables another thread to process the data. When the threads invoked by AceClientCheck finish, the developer-defined function is called (if one has been supplied).

Parameters

	Parameter	Description
[in]	SdiHandle	The value of a handle originally assigned by a call to AceInit.
[in]	appCallback	A pointer to a function written by the developer. This function returns nothing. The handle to the data area specific to the authentication process is passed to this function. A NULL value can be passed instead of a valid function pointer.

Return Values

The value returned by this function is one of the following

Returned Value	Description
ACE_PROCESSING	Normal return.
ACE_ERR_INVALID_HANDLE	AceClientCheck could not locate the data associated with the handle.
ACE_UNDEFINED_PASSCODE	The passcode or credential has yet to be set for the authentication request.

Returned Value	Description
ACE_UNDEFINED_USERNAME	The user name is yet to be set for the authentication request.
ACE_UNDEFINED_CLIENTADDR	The client address is yet to be set for the authentication request.
ACE_TOO_MANY_CALLERS	An attempt was made to use a handle while a call was already in progress. This happens if a multithreaded application uses the same handle from multiple threads at the same time. This return code is not valid in this version.

Error Handling

To handle errors appropriately, use the value returned by this function at a decision point in your code. A successful return allows processing to continue.

Checking the Results

A caller can verify the result of the authentication operation by calling the AceGetAuthenticationStatus function once the requested operation is completed.

Result	Description
ACM_OK	The user was successfully authenticated. Use the AceGetShell function to access the shell field.
ACM_ACCESS_DENIED	The user failed authentication.
ACM_NEXT_CODE_REQUIRED	Next tokencode required. Use AceSetNextPasscode or AceSetNextCredential and AceSendNextPasscode to complete the transaction. The AceClientCheck API must be called within 140 seconds if using Authentication Manager 6.1.2, or 120 seconds if using Authentication Manager 7.1. If using Authentication Manager 8.0, the timeout is 180 seconds.
ACM_NEW_PIN_REQUIRED	New PIN required. You can use the AceGetxxx function to access the PIN limits. Use AceSetPin and AceSendPin, or AceCancelPin to complete the transaction. These APIs must be called within 140 seconds if using Authentication Manager 6.1.2, or 120 seconds if using Authentication Manager 7.1. If using Authentication Manager 8.0, the timeout is 180 seconds.

Requirements

Minimum supported Authentication Manager Server Version	6.1 or later
Category	Asynchronous API
Header	acexport.h
Static Library	UNIX: libaceclnt.a
Shared Library	Windows: • aceclnt.dll • aceclnt_tcp.dll • sdmsg.dll • xerces-c_3_1_vc80.dll UNIX: • libaceclnt.so • libaceclnt_tcp.so • libxerces-c-3.1.so AIX: • libaceclnt.so **Note:** Rename to libaceclnt.a. For more information, see “Using Dynamic Libraries or Static Libraries in UNIX” on page 38. • libaceclnt_tcp.so • libxerces-c-3.1.a

AceClose

Description

```
int WINAPI AceClose(
  SDI_HANDLE SdiHandle,
  void (WINAPI*appCallback)(SDI_HANDLE SdiHandle));
```

The AceClose function closes the socket and releases the data associated with the authentication process specified by SdiHandle. You must call this function after attempting to authenticate the user, regardless of whether the authentication was successful. The one exception is if ACE_PROCESSING was never returned from a call to AceInit.

Architecture

This asynchronous function returns immediately with a status value. If the status is ACE_PROCESSING, threads are enabled to complete processing the data.

The AceClose function attempts to find the data associated with this specific authentication process through the unique handle value. If the data is found, AceClose calls another thread to process the data. When the threads invoked by AceClose finish, the developer-defined function is called (if one has been supplied).

Parameters

	Parameter	Description
[in]	SdiHandle	The value of a handle originally assigned by a call to AceInit.
[in]	appCallback	A pointer to a function written by the developer. This function returns nothing. The handle to the data area specific to the authentication process is passed to this function. A NULL value can be passed instead of a valid function pointer.

Return Values

The value returned by this function is one of the following.

Returned Value	Description
ACE_PROCESSING	Normal return.
ACE_ERR_INVALID_HANDLE	AceClose could not locate the data associated with the handle.

Note: The handle used in the AceClose call cannot be accessed after the callback has been invoked. As AceGetUserData requires a valid handle, an attempt to access UserData for a specific authentication request fails after AceClose has released the resources associated with that request. However, the callback function supplied to AceClose is able to access the data. If you want to dispose the data, do so before calling AceClose or during the callback.

Error Handling

To handle errors appropriately, use the value returned by this function at a decision point in your code. A successful return allows processing to continue.

Checking the Results

If the caller supplied a callback function, it can verify the result of the authentication operation by calling the AceGetAuthenticationStatus function. The following table lists the possible status code values.

ACM_OK	The authentication handle was disposed of successfully.

Requirements

Minimum supported Authentication Manager Server Version	6.1 or later
Category	Asynchronous API
Header	acexport.h
Static Library	UNIX: libaceclnt.a
Shared Library	Windows: • aceclnt.dll • aceclnt_tcp.dll • sdmsg.dll • xerces-c_3_1_vc80.dll UNIX: • libaceclnt.so • libaceclnt_tcp.so • libxerces-c-3.1.so AIX: • libaceclnt.so **Note:** Rename to libaceclnt.a. For more information, see "Using Dynamic Libraries or Static Libraries in UNIX" on page 38. • libaceclnt_tcp.so • libxerces-c-3.1.a

AceCloseAuth

Description

```
SD_ERROR WINAPI AceCloseAuth(
  SDI_HANDLE SdiHandle)
```

The AceCloseAuth function closes down an authentication context and releases any memory that was allocated on its behalf. It can be called any time after a successful return from AceStartAuth for the particular authentication context specified by the SDI_HANDLE value. This function is the final step in the synchronous API model.

Note: If you use RSA Authentication Agent API 6.0 with earlier (pre-5.0) agents, you do not have to change your use of the AceStartAuth, AceContinueAuth, and AceCloseAuth functions to support two-step authentication. Two-step authentication is automatically performed within these three functions.

Architecture

The caller of this synchronous function must supply as the first argument an SDI_HANDLE value that was previously set in the call to AceStartAuth.

Parameters

	Parameter	Description
[in]	hdl	An SDI_HANDLE associated with the authentication context.

Return Values

If the return value is ACM_OK, it is successful. Otherwise, the return value is one of the errors returned by the AceClose asynchronous API call.

Note: The handle used in the AceCloseAuth call is invalid after the function returns. To retrieve the result of the authentication, call AceGetAuthenticationStatus before calling AceCloseAuth. If AceGetAuthenticationStatus is mistakenly called immediately after AceCloseAuth, the process returns the status of the close, not the authentication.

Error Handling

To handle errors appropriately, use the value returned by this function at a decision point in your code. A successful return allows processing to continue.

Requirements

Minimum supported Authentication Manager Server Version	6.1 or later
Category	Synchronous API
Header	acexport.h
Static Library	UNIX: libaceclnt.a
Shared Library	Windows: • aceclnt.dll • aceclnt_tcp.dll • sdmsg.dll • xerces-c_3_1_vc80.dll UNIX: • libaceclnt.so • libaceclnt_tcp.so • libxerces-c-3.1.so AIX: • libaceclnt.so **Note:** Rename to libaceclnt.a. For more information, see “Using Dynamic Libraries or Static Libraries in UNIX” on page 38. • libaceclnt_tcp.so • libxerces-c-3.1.a

AceContinueAuth

Description

```
SD_ERROR WINAPI AceContinueAuth(
  SDI_HANDLE SdiHandle,
  char *resp,
  SD_I32 respLen,
  SD_BOOL *moreData,
  SD_BOOL *echoFlag,
  SD_I32 *respTimeout,
  SD_I32 *nextRespLen,
  char *promptStr,
  SD_I32 *promptStrLen)
```

The AceContinueAuth function supplies the next value needed by the authentication context. It is the middle step in the synchronous API model that begins with AceStartAuth and ends with AceCloseAuth.

The values of the prompts returned by this function are stored in a message catalog that can be modified by an agent developer. For more information, see Appendix C, "Modifying the Message Catalog."

Note: If you use version 6.0 of the RSA Authentication API with earlier (pre-5.0) agents, you do not have to change your use of the AceStartAuth, AceContinueAuth, and AceCloseAuth functions to support two-step authentication. Two-step authentication is automatically performed within these three functions.

Architecture

Your code must continue to supply data to the authentication context through this function as long as the moreData argument is set to True. The promptStr argument contains the string used to display the next response message to the user attempting authentication. This string, which is set by the API, contains either a prompt for the user, for example, "Enter next tokencode:", or a statement intended to provide status information to the user, for example, "Access denied."

Note: The return value of AceContinueAuth indicates the success or failure of the function, not of the authentication in progress. To get the actual authentication status, use the AceGetAuthenticationStatus function with the handle returned from AceStartAuth.

Parameters

	Parameter	Description
[in]	hdl	The SDI_HANDLE whose value is filled in by the AceStartAuth function.
[in]	resp	A pointer to the string containing the response value.
[in]	respLen	An integer containing the length of the response string.
[out]	moreData	A flag that indicates if more data is needed by the authentication context.
[out]	noEcho	A flag that hints the developer as to whether the next expected response is echoed to the screen.
[out]	respTimeout	A hint to the developer about how long to display the next prompt string.
[out]	nextRespLen	Indicates the maximum number of bytes of data expected in the next developer-supplied response.
[out]	promptStr	Developer-supplied character array to be filled in with the string to be used as the next message displayed to the user.
[in, out]	promptStrLen	Initially set to the size of the developer-supplied storage for the promptStr, the value becomes the length of the filled-in promptStr string.

Return Values

The return value indicates whether the function was successful in copying the data pointed to by the response. An ACM_OK value is returned on success, and an error value is returned upon failure. The following table lists typical return values.

Return Value	Description
ACM_OK	The call completed successfully. Use AceGetAuthenticationStatus to determine the success or failure of the authentication.
ACE_INVALID_ARG	Could not set the data pointed to by response as the next expected data value.
ACE_ERR_INVALID_HANDLE	The handle value is invalid.

Return Value	Description
ACE_NOT_ENOUGH_STORAGE	The size of the developer-supplied promptStr is too small. PromptStrLen holds the required size of the PromptStr. PromptStr reads "Program error: buffer too small." If the developer-supplied array is too small even for this error message, the error message is truncated to promptLen supplied.
ACE_TOO_MANY_CALLERS	An attempt was made to use a handle while a call was already in progress. This happens if a multithreaded application uses the same handle from multiple threads at the same time. This return code is not valid in this version.

Error Handling

Your code must close down the authentication context using AceCloseAuth regardless of whether this function was successful.

Checking the Results

Calling the AceGetAuthenticationStatus function allows the caller to verify the actual status of the authentication operations. These status codes are defined in the description of the AceGetAuthenticationStatus function.

Requirements

Minimum supported Authentication Manager Server Version	6.1 or later
Category	Synchronous API
Header	acexport.h
Static Library	UNIX: libaceclnt.a

Minimum supported Authentication Manager Server Version	6.1 or later
Shared Library	Windows: • aceclnt.dll • aceclnt_tcp.dll • sdmsg.dll • xerces-c_3_1_vc80.dll UNIX: • libaceclnt.so • libaceclnt_tcp.so • libxerces-c-3.1.so AIX: • libaceclnt.so **Note:** Rename to libaceclnt.a. For more information, see “Using Dynamic Libraries or Static Libraries in UNIX” on page 38. • libaceclnt_tcp.so • libxerces-c-3.1.a

AceDisableNodeSecretCache

Note: This function is not supported in this release of the API, and will always return success.

Description

```
void AceDisableNodeSecretCache()
```

This synchronous function disables the caching of node secret. For every authentication request, the node secret is read from the node secret file. This enables agents to change the node secret file while the authentication library is loaded. If your agent processes multiple authentication requests, this API makes your agent very slow.

Note: RSA recommends not to disable the node secret cache option in performance intensive agents.

Requirements

Minimum supported Authentication Manager Server Version	6.1 or later
Category	Synchronous API
Header	acexport.h
Static Library	UNIX: libaceclnt.a
Shared Library	Windows: aceclnt.dll UNIX: libaceclnt.so libaceclnt_tcp.so

AceGetAlphanumeric

Description

```
int WINAPI AceGetAlphanumeric(
  SDI_HANDLE SdiHandle,
  char *val)
```

The AceGetAlphanumeric function determines whether the PIN of the authenticating user can have alphanumeric characters.

It is helpful to retrieve this value in two cases: first, when the result code from AceCheck is ACM_NEW_PIN_REQUIRED; and second, when the agent is designed to validate that the new PIN entered by the user conforms to the required PIN characteristics set by the RSA Authentication Manager administrator.

Any such check done by the agent is a convenience to the user. Authentication Manager checks for the correctness of the PIN.

Note: To obtain all the PIN-related parameters at once, use AceGetPinParams.

Architecture

This synchronous function fills in the second argument passed to it with a value that indicates whether or not a user's new PIN can have alphanumeric characters.

Parameters

	Parameter	Description
[in]	hdl	The value of a handle originally assigned by a call to AceInit.
[out]	val	A pointer to a character that is assigned the alphanumeric flag value.

Val Data Value	Description
0	The user's PIN can have only numeric values.
1	The user's PIN can have alphabetic or numeric values.

Return Values

The function returns ACE_SUCCESS if the handle to the authentication data was found and the alphanumeric value was filled in by the function. The following table lists the possible values.

Return Value	Description
ACE_INVALID_ARG	Do not have write access to the parameter val.
ACE_TOO_MANY_CALLERS	An attempt was made to use a handle while a call was already in progress. This happens if a multithreaded application uses the same handle from multiple threads at the same time. This return code is not valid in this version.

Error Handling

The function returns a value of ACE_ERR_INVALID_HANDLE if the handle to the authentication data cannot be found.

Requirements

Minimum supported Authentication Manager Server Version	6.1 or later
Category	Synchronous API
Header	acexport.h
Static Library	UNIX: libaceclnt.a
Shared Library	Windows: • aceclnt.dll • aceclnt_tcp.dll • sdmsg.dll • xerces-c_3_1_vc80.dll UNIX: • libaceclnt.so • libaceclnt_tcp.so • libxerces-c-3.1.so AIX: • libaceclnt.so **Note:** Rename to libaceclnt.a. For more information, see "Using Dynamic Libraries or Static Libraries in UNIX" on page 38. • libaceclnt_tcp.so • libxerces-c-3.1.a

AceGetAuthAttr

Description

```
int WINAPI AceGetAuthAttr (
  SDI_HANDLE SdiHandle,
  RSA_AUTH_GET_ATTR authAttribute,
  void *attrValue,
  SD_U32 *attrSize )
```

The AceGetAuthAttr function retrieves a particular authentication attribute from the RSA Authentication Manager. The definitions of the attributes and associated data are described in the header file **acexport.h**. The function can retrieve attributes after a synchronous or asynchronous authentication request has completed successfully. For example, AceGetAuthAttr retrieves authentication attributes in Next Tokencode mode and New PIN mode.

AceGetAuthAttr is part of the Generic Credentials API. For more information, see "Support for EAP 32 and the Generic Credential API" on page 13.

Note: Although the functionality for AceGetAuthAttr has changed from previous releases of the RSA Authentication Agent API for C, the interface remains unchanged.

Parameters

	Parameter	Description
[in]	SdiHandle	Handle to the current authentication session, which SDInit created.
[in]	authAttribute	The identifier of the requested authentication attribute.
[out]	attrValue	Pointer to the buffer that will store the authentication attribute.
[in, out]	attrSize	Pointer to the buffer size.

Note: To obtain the required size of the output buffer **attrValue**, set the **attrSize** value to 0 and call the AceGetAuthAttr function. When the function returns ACE_NOT_ENOUGH_STORAGE, attrSize points to the required size. Use the required size to allocate the output buffer. Call the function again to receive the attribute.

Return Values

Return Value	Description
ACE_SUCCESS	Function has successfully retrieved the authentication attribute.
ACE_NOT_ENOUGH_STORAGE	Caller buffer is too small. **Note**: attrSize points to the required size of the output buffer.
ACE_INVALID_ARG	Function cannot recognize the requested authentication attribute.
ACE_TOO_MANY_CALLERS	An attempt was made to use a handle while a call was already in progress. This happens if a multithreaded application uses the same handle from multiple threads at the same time. This return code is not valid in this version.

Important: There is a design constraint on the use of SDI_HANDLE. An asynchronous application of the API must avoid concurrent use of the same SDI_HANDLE. Multiple threads using the same SDI_HANDLE return incorrect results.

Error Handling

To handle errors appropriately, use the value returned by this function at a decision point in your code. A successful return allows processing to continue.

RSA_AUTH_GET_ATTR

Value	Description
RSA_AUTH_GET_ATTR_RADIUS_PROFILE	Get Radius profile
RSA_AUTH_GET_ATTR_RADIUS_EXTENSIONS	Get Radius user extensions
RSA_AUTH_GET_ATTR_EAP32_KEYS	Get EAP Keys
RSA_AUTH_GET_ATTR_EAP32_PEPPER	Get EAP data
RSA_AUTH_GET_ATTR_EAP32_MAC	EAP32 MAC Address

Note: The below tags are not supported in this version and they will be ignored. The API will always return success for these tags:
RSA_AUTH_GET_ATTR_PIN_LENGTH:
RSA_AUTH_GET_ATTR_AGENT_TYPE:
RSA_AUTH_GET_ATTR_AUTHENTICATOR_ID:
RSA_AUTH_GET_ATTR_AUTH_HASH:
RSA_AUTH_GET_ATTR_DOMAIN_CLIENT_VERSION:

Requirements

Minimum supported Authentication Manager Server Version	6.1 or later
Category	Synchronous API
Header	acexport.h
Static Library	UNIX: libaceclnt.a
Shared Library	Windows: • aceclnt.dll • aceclnt_tcp.dll • sdmsg.dll • xerces-c_3_1_vc80.dll UNIX: • libaceclnt.so • libaceclnt_tcp.so • libxerces-c-3.1.so AIX: • libaceclnt.so **Note:** Rename to libaceclnt.a. For more information, see “Using Dynamic Libraries or Static Libraries in UNIX” on page 38. • libaceclnt_tcp.so • libxerces-c-3.1.a

AceGetAuthenticationStatus

Description

```
int WINAPI AceGetAuthenticationStatus(
  SDI_HANDLE SdiHandle,
  INT32BIT *val)
```

The AceGetAuthenticationStatus function determines the status of the authentication request at each step in the process. This function is useful as part of a callback function in asynchronous APIs.

Architecture

This synchronous function fills in the current value for the result code associated with the authentication request data.

Parameters

	Parameter	Description
[in]	hdl	The value of a handle originally assigned by a call to AceInit.
[out]	val	A 32-bit integer pointer to data that holds the result code value.

Return Values

The function returns ACE_SUCCESS if the handle to the authentication data was found and the result code was filled in by the function. A value of ACE_ERR_INVALID_HANDLE is returned if the handle to the authentication data cannot be found.

The following table lists some of the more common values and meanings for the result codes.

Return Value	Description
ACM_OK	The meaning of ACM_OK depends on the function that caused it to be set as a result code. For details, see the table in Appendix A, "Return Values and Result Codes."
ACM_NEXT_CODE_REQUIRED	Another passcode must be supplied before authentication can be granted.
ACM_ACCESS_DENIED	The user's authentication failed.
ACM_NEW_PIN_REQUIRED	The token is in New PIN mode.

Return Value	Description
ACM_NEW_PIN_ACCEPTED	The Authentication Manager has accepted a new PIN. The user is now required to authenticate with the new PIN.
ACM_NEW_PIN_REJECTED	The Authentication Manager rejected the new PIN. The PIN might not have matched the parameters set in the return from AceGetPinParams.
ACM_NEXT_CODE_BAD	The passcode supplied to the Authentication Manager is not valid.
ACE_INVALID_ARG	Do not have write access to the parameter val.

Important: There is a design constraint on the use of SDI_HANDLE. An asynchronous application of the API must avoid concurrent use of the same SDI_HANDLE. Multiple threads using the same SDI_HANDLE return incorrect results.

Error Handling

To handle errors appropriately, use the values set by this function at decision points in your code.

Requirements

Minimum supported Authentication Manager Server Version	6.1 or later
Category	Synchronous API
Header	acexport.h
Static Library	UNIX: libaceclnt.a
Shared Library	Windows: • aceclnt.dll • aceclnt_tcp.dll • sdmsg.dll • xerces-c_3_1_vc80.dll UNIX: • libaceclnt.so • libaceclnt_tcp.so • libxerces-c-3.1.so AIX: • libaceclnt.so **Note:** Rename to libaceclnt.a. For more information, see “Using Dynamic Libraries or Static Libraries in UNIX” on page 38. • libaceclnt_tcp.so • libxerces-c-3.1.a

AceGetDAuthenticationStatus

Note: This function is not supported in this release of the API. It will always return success.

This function returns the status of offline authentication. The definition of the status is described in the header file, **da_svc_api.h**.

Note: In this release of the API, this function is for the Windows platform only. As this function will not be supported in later releases of the API, RSA recommends that you use the AceGetDAAuthData function instead.

Description

```
int WINAPI AceGetDAuthenticationStatus (
   SDI_HANDLE   hdl,
   INT32BIT *   val)
```

To call the AceGetDAuthenticationStatus function, the API must have already established communication with the offline service (through a call to SD_Init, or to SD_InitEx by setting bSupportOA to SD_TRUE). In addition, an offline authentication must have been attempted before this call.

Parameters

	Parameter	Description
[in]	hdl	The value of a handle originally assigned by a call to SD_Init or SD_InitEx.
[out]	val	A pointer to a signed 32-bit value that is assigned the result of the offline authentication attempt.

Return Values

Return Value	Description
ACE_SUCCESS	Indicates that the status has been successfully retrieved.
val	Indicates failure to retrieve the status.
ACE_INVALID_ARG	Do not have write access to the parameter val.

Requirements

Minimum supported Authentication Manager Server Version	6.1 or later
Category	Synchronous API
Header	acexport.h
Static Library	Windows: libaceclnt.lib

Minimum supported Authentication Manager Server Version	6.1 or later
Shared Library	Windows: • aceclnt.dll • aceclnt_tcp.dll • sdmsg.dll • xerces-c_3_1_vc80.dll UNIX: • libaceclnt.so • libaceclnt_tcp.so • libxerces-c-3.1.so AIX: • libaceclnt.so **Note:** Rename to libaceclnt.a. For more information, see “Using Dynamic Libraries or Static Libraries in UNIX” on page 38. • libaceclnt_tcp.so • libxerces-c-3.1.a

AceGetDAAuthData

Note: This function is not supported in this release of the API,

This function returns offline authentication attributes. The definitions of the attributes and associated data are described in the header files, **da_svc_api.h** and **acexport.h**.

Note: In this release of the API, this function is for the Windows platform only.

Description

```
int WINAPI AceGetDAAuthData (
    SDI_HANDLE          hdl,
    RSA_DA_ATTR         attribute,
    void *              pattrBuf,
    SD_U32 *            size)
```

To call the AceGetDAAuthData function, the API must have already established communication with the offline service (through a call to SD_Init, or to SD_InitEx by setting bSupportOA to SD_TRUE). In addition, certain offline operations must have been attempted before this call, depending on the type of attributes being requested.

Parameters

	Parameter	Description
[in]	hdl	The value of a handle originally assigned by a call to SD_Init or SD_InitEx.
[in]	attribute	The type of the attribute being requested. The values are defined in both **acexport.h** and **da_svc_api.h**.
[in]	pAttrBuf	A pointer to a buffer that has been allocated by the caller to receive a copy of the attribute data.
[in, out]	size	A pointer to an unsigned 32-bit value that initially specifies the length of the output buffer pointed to by pAttrBuf, and is assigned the size of the returned data in bytes. The recommended and maximum sizes of the data vary depending on the attribute type.

Return Values

Return Value	Description
ACE_SUCCESS	Indicates that the data has been successfully retrieved.
Other errors	Indicates failure to retrieve the data.
ACE_ERR_INVALID_HANDLE	Invalid handle. Failure to locate the data associated with this handle.
ACE_INVALID_ARG	Input parameter pAttrBuf or size is not valid.
ACE_TOO_MANY_CALLERS	An attempt was made to use a handle while a call was already in progress. This happens if a multithreaded application uses the same handle from multiple threads at the same time. This return code is not valid in this version.
ACE_NOT_ENOUGH_STORAGE	When the function returns ACE_NOT_ENOUGH_STORAGE, size points to the required length for pAttrBuf buffer. Use this size to allocate the pAttrBuf buffer. Call the function again to receive the Auth data.

RSA_DA_ATTR

Value	Description
RSA_DA_ATTR_AUTH_TIME	The time the auth successfully matched.
RSA_DA_ATTR_POA	The hash derived Proof Of Authentication Key associated with the successful authentication
RSA_DA_ATTR_EAC_EXP	If the authentication was an EAC auth, this returns the date and time the EAC code used will expire.
RSA_DA_ATTR_TICKET	A 24-character long download ticket number.
RSA_DA_ATTR_LOGIN_PW	The user's password recovered as a result of a successful disconnected authentication.
RSA_DA_ATTR_SERVER_LIST	Provides a list of all the servers.
RSA_DA_ATTR_DOWLOAD_STATUS	Provides the download status of day files.
RSA_DA_ATTR_TOKEN_SERIAL	For obtaining token serial number.

Value	Description
RSA_DA_ATTR_PASSWORD_CHANGED	Provides the status of change in Windows passwords.
RSA_DA_ATTR_POLICY_CHANGED	User policy status
RSA_DA_ATTR_DADAYS	Offline authentication data days.
RSA_DA_ATTR_LPIENABLED	Provides the status of login password integration.
RSA_DA_ATTR_DAENABLED	Provides status of Offline Authentication flag in user policy.
RSA_DA_ATTR_SALT	Provides the current salt value.
RSA_DA_ATTR_OA_VERSION	Obtains OA version
RSA_DA_ATTR_PIN_LENGTH	Gives length of the PIN

Requirements

Minimum supported Authentication Manager Server Version	6.1 or later
Category	Synchronous API
Header	acexport.h
Static Library	Windows: libaceclnt.lib
Shared Library	Windows: • aceclnt.dll • aceclnt_tcp.dll • sdmsg.dll • xerces-c_3_1_vc80.dll UNIX: • libaceclnt.so • libaceclnt_tcp.so • libxerces-c-3.1.so AIX: • libaceclnt.so **Note:** Rename to libaceclnt.a. For more information, see "Using Dynamic Libraries or Static Libraries in UNIX" on page 38. • libaceclnt_tcp.so • libxerces-c-3.1.a

AceGetIterCountPolicy

Description

```
int WINAPI AceGetIterCountPolicy (
  SDI_HANDLE hdl,
  INT32BIT *min,
  INT32BIT *max )
```

The AceGetIterCountPolicy function returns the EAP 32 iteration count policy.

AceGetIterCountPolicy is part of the Generic Credentials API. For more information, see "Support for EAP 32 and the Generic Credential API" on page 13.

Parameters

	Parameter	Description
[in]	hdl	Handle to the current authentication session, which SDInit created.
[out]	min	Pointer to a 32-bit integer that represents the minimum EAP 32 iteration count.
[out]	max	Pointer to a 32-bit integer that represents the maximum EAP 32 iteration count.

Return Values

Return Value	Description
ACE_SUCCESS	Function has successfully retrieved the authentication attribute.
ACE_ERR_INVALID_HANDLE	Invalid handle, could not locate the data associated with this handle.
ACE_TOO_MANY_CALLERS	An attempt was made to use a handle while a call was already in progress. This happens if a multithreaded application uses the same handle from multiple threads at the same time. This return code is not valid in this version.
ACE_INVALID_ARG	Input parameter max or min is not valid.

Important: There is a design constraint on the use of SDI_HANDLE. An asynchronous application of the API must avoid concurrent use of the same SDI_HANDLE. Multiple threads using the same SDI_HANDLE return incorrect results.

Error Handling

To handle errors appropriately, use the value returned by AceGetIterCountPolicy at a decision point in your code. A successful return allows processing to continue.

Requirements

Minimum supported Authentication Manager Server Version	6.1 or later
Category	Synchronous API
Header	acexport.h
Static Library	UNIX: libaceclnt.a
Shared Library	Windows: • aceclnt.dll • aceclnt_tcp.dll • sdmsg.dll • xerces-c_3_1_vc80.dll UNIX: • libaceclnt.so • libaceclnt_tcp.so • libxerces-c-3.1.so AIX: • libaceclnt.so **Note:** Rename to libaceclnt.a. For more information, see “Using Dynamic Libraries or Static Libraries in UNIX” on page 38. • libaceclnt_tcp.so • libxerces-c-3.1.a

AceGetLoginPW

Note: This function is not supported in this version of the API and will always return success.

This function retrieves the user's logon password from the Authentication Manager database after a successful authentication.

Description

```
int WINAPI AceGetLoginPW (
   SDI_HANDLE          hdl,
   SD_CHAR *val,
   SD_U32 *size)
```

Before calling the AceGetLoginPW function, a successful authentication is required. In addition, you must configure Authentication Manager to allow password integration.

Parameters

	Parameter	Description
[in]	hdl	The value of a handle originally assigned by a call to SD_Init or SD_InitEx.
[out]	val	A pointer to a character buffer that has been allocated by the caller and receives a copy of the password. Note that the password is not NULL-terminated. The pointer to the size contains the length of the password.
[in, out]	size	A pointer to an unsigned 32-bit value that initially specifies the length of the output buffer pointed to by val, and that is assigned the length of the password. The maximum size of the password is 128 bytes.

Note: To obtain the required size of the output buffer pchPWBuffer, set the length to which pu32PWLen points to 0 and call this AceGetLoginPW. When the function returns ACE_NOT_ENOUGH_STORAGE, pu32PWLen points to the required size. Use this size to allocate the output buffer. Call the function again to receive the password.

Return Values

Return Value	Description
ACE_SUCCESS	The password has been successfully retrieved.
ACE_NOT_ENOUGH_STORAGE	Caller must supply a larger buffer; pu32PWLen points to the length of the required size.
ACE_ERR_INVALID_HANDLE	Invalid handle, could not locate the data associated with this handle.
ACE_DA_INVALID_PASSWORD	Fetching Obfuscated Password failed.
ACE_TOO_MANY_CALLERS	An attempt was made to use a handle while a call was already in progress. This happens if a multithreaded application uses the same handle from multiple threads at the same time. This return code is not valid in this version.
ACE_INVALID_ARG	Input parameter val or size is not valid.
Other errors	Indicates failure to retrieve the password.

Requirements

Minimum supported Authentication Manager Server Version	6.1 or later
Category	Synchronous API
Header	acexport.h
Static Library	UNIX: libaceclnt.a

Minimum supported Authentication Manager Server Version	6.1 or later
Shared Library	Windows: • aceclnt.dll • aceclnt_tcp.dll • sdmsg.dll • xerces-c_3_1_vc80.dll UNIX: • libaceclnt.so • libaceclnt_tcp.so • libxerces-c-3.1.so AIX: • libaceclnt.so **Note:** Rename to libaceclnt.a. For more information, see "Using Dynamic Libraries or Static Libraries in UNIX" on page 38. • libaceclnt_tcp.so • libxerces-c-3.1.a

AceGetMaxPinLen

Description

```
int WINAPI AceGetMaxPinLen(
  SDI_HANDLE hdl,
  char *val)
```

The AceGetMaxPinLen function determines the value of the maximum PIN length allowed by the Authentication Manager.

It is helpful to retrieve this value in two cases: first, when the result code from AceCheck is ACM_NEW_PIN_REQUIRED; and second, when the agent is designed to validate that the new PIN entered by the user conforms to the required PIN characteristics set by the RSA Authentication Manager administrator.

Any such checking done by the agent is simply as a convenience to the user. The PIN is fully checked for correctness by the Authentication Manager.

Note: To obtain all the PIN-related parameters at once, use AceGetPinParams.

Architecture

This function is synchronous. It uses the second argument passed to it to fill in the current value for the maximum PIN length associated with the authentication request data.

Parameters

	Parameter	Description
[in]	hdl	The value of a handle originally assigned by a call to AceInit.
[out]	val	A pointer to a character that will be assigned the maximum PIN length value.

Note: AceGetMaxPinLen must be called only when a user is in New PIN mode. It helps agents to validate that the new PIN entered by the user conforms to the required PIN characteristics set by the Authentication Manager administrator.

Return Values

The value returned by this function is one of the following.

Return Value	Description
ACE_SUCCESS	Handle to the authentication data was found and the maximum PIN length currently allowed by the RSA Authentication Manager was filled in by the function.

Return Value	Description
ACE_ERR_INVALID_HANDLE	Invalid handle, could not locate the data associated with this handle.
ACE_TOO_MANY_CALLERS	An attempt was made to use a handle while a call was already in progress. This happens if a multithreaded application uses the same handle from multiple threads at the same time. This return code is not valid in this version.
ACE_INVALID_ARG	Input parameter value is not valid.

Error Handling

A value of **ACE_ERR_INVALID_HANDLE** is returned if the handle to the authentication data cannot be found.

Requirements

Minimum supported Authentication Manager Server Version	6.1 or later
Category	Synchronous API
Header	acexport.h
Static Library	UNIX: libaceclnt.a
Shared Library	Windows: • aceclnt.dll • aceclnt_tcp.dll • sdmsg.dll • xerces-c_3_1_vc80.dll UNIX: • libaceclnt.so • libaceclnt_tcp.so • libxerces-c-3.1.so AIX: • libaceclnt.so **Note:** Rename to libaceclnt.a. For more information, see "Using Dynamic Libraries or Static Libraries in UNIX" on page 38. • libaceclnt_tcp.so • libxerces-c-3.1.a

AceGetMinPinLen

Description

```
int WINAPI AceGetMinPinLen(
  SDI_HANDLE hdl,
  char *val)
```

The AceGetMinPinLen function determines the value of the minimum PIN length allowed by theRSA Authentication Manager. It is helpful to retrieve this value when the result code from AceCheck is ACM_NEW_PIN_REQUIRED and the agent is designed to validate that the new PIN entered by the user conforms to the required PIN characteristics set by the RSA Authentication Manager administrator. Any such check done by the agent is simply as a convenience to the user. RSA Authentication Manager checks the PIN for correctness.

Note: To obtain all the PIN-related parameters at once, use AceGetPinParams.

Architecture

This synchronous function uses the second argument passed to it to fill in the current value for the minimum PIN length associated with the authentication request data.

Parameters

	Parameter	Description
[in]	hdl	The value of a handle originally assigned by a call to AceInit.
[out]	val	A pointer to a character that is assigned the minimum PIN length value.

Note: AceGetMinPinLen must be called only when a user is in New PIN mode. It helps agents to validate that the new PIN entered by the user conforms to the required PIN characteristics set by the Authentication Manager administrator.

Return Values

The value returned by this function is one of the following.

Return Value	Description
ACE_SUCCESS	Handle to the authentication data was found and the minimum PIN length currently allowed by the RSA Authentication Manager was filled in by the function.
ACE_ERR_INVALID_HANDLE	Invalid handle, could not locate the data associated with this handle.

Return Value	Description
ACE_TOO_MANY_CALLERS	An attempt was made to use a handle while a call was already in progress. This happens if a multithreaded application uses the same handle from multiple threads at the same time. This return code is not valid in this version.
ACE_INVALID_ARG	Input parameter value is not valid.

Error Handling

A value of ACE_ERR_INVALID_HANDLE is returned if the handle to the authentication data cannot be found.

Requirements

Minimum supported Authentication Manager Server Version	6.1 or later
Category	Synchronous API
Header	acexport.h
Static Library	UNIX: libaceclnt.a
Shared Library	Windows: • aceclnt.dll • aceclnt_tcp.dll • sdmsg.dll • xerces-c_3_1_vc80.dll UNIX: • libaceclnt.so • libaceclnt_tcp.so • libxerces-c-3.1.so AIX: • libaceclnt.so **Note:** Rename to libaceclnt.a. For more information, see "Using Dynamic Libraries or Static Libraries in UNIX" on page 38. • libaceclnt_tcp.so • libxerces-c-3.1.a

AceGetPepperPolicy

Description

```
int WINAPI AceGetPepperPolicy (
  SDI_HANDLE hdl,
  char *min,
  char *max )
```

The AceGetPepperPolicy function returns the minimum and maximum EAP 32 pepper lengths (in bytes).

The EAP 32 protocol uses an encrypted string known as a "pepper value," which is stored in the Authentication Manager database, to protect one-time passwords. The minimum default pepper length that the Authentication Manager accepts is 4 bytes and the maximum default is 12 bytes.

Note: RSA recommends that you use the default values. Contact RSA Security Customer Support before you change any of these values.

AceGetPepperPolicy is part of the Generic Credentials API. For more information, see "Support for EAP 32 and the Generic Credential API" on page 13.

Parameters

	Parameter	Description
[in]	hdl	Handle to the current authentication session, which SD_Init created.
[out]	min	Pointer to an 8-bit integer that represents the minimum EAP 32 pepper length.
[out]	max	Pointer to an 8-bit integer that represents the maximum EAP 32 pepper length.

Return Values

The value returned by this function is one of the following.

Return Value	Description
ACE_SUCCESS	The function has successfully retrieved the authentication attribute.
ACE_ERR_INVALID_HANDLE	Invalid handle, could not locate the data associated with this handle.

Return Value	Description
ACE_TOO_MANY_CALLERS	An attempt was made to use a handle while a call was already in progress. This happens if a multithreaded application uses the same handle from multiple threads at the same time. This return code is not valid in this version.
ACE_INVALID_ARG	Input parameter min or max is not valid.

Important: There is a design constraint on the use of SDI_HANDLE. An asynchronous application of the API must avoid concurrent use of the same SDI_HANDLE. Multiple threads using the same SDI_HANDLE return incorrect results.

Error Handling

To handle errors appropriately, use the value returned by this function at a decision point in your code. A successful return allows processing to continue.

Requirements

Minimum supported Authentication Manager Server Version	6.1 or later
Category	Synchronous API
Header	acexport.h
Static Library	UNIX: libaceclnt.a
Shared Library	Windows: • aceclnt.dll • aceclnt_tcp.dll • sdmsg.dll • xerces-c_3_1_vc80.dll UNIX: • libaceclnt.so • libaceclnt_tcp.so • libxerces-c-3.1.so AIX: • libaceclnt.so **Note:** Rename to libaceclnt.a. For more information, see “Using Dynamic Libraries or Static Libraries in UNIX” on page 38. • libaceclnt_tcp.so • libxerces-c-3.1.a

AceGetPinParams

Description

```
int WINAPI AceGetPinParams (
  SDI_HANDLE hdl,
  SD_PIN *pSdPin)
```

The AceGetPinParams function obtains all of the PIN-related parameters in a single call. The definitions of the attributes and associated data are described in the header file **acexport.h**. It is helpful to retrieve this value when the result code from AceCheck is ACM_NEW_PIN_REQUIRED and the agent is designed to validate that the new PIN entered by the user conforms to the required PIN characteristics set by the RSA Authentication Manager administrator. Any such checking done by the agent is simply as a convenience to the user. The PIN is fully checked for correctness by the RSA Authentication Manager.

Architecture

The caller of this synchronous function must supply, as the second argument, a pointer to a structure of type SD_PIN, into which the function copies the PIN parameters. AceGetPinParams does not allocate storage for the data on its own.

Parameters

	Parameter	Description
[in]	hdl	The value of a handle originally assigned by a call to AceInit.
[out]	pSdPin	A pointer to a structure of type SD_PIN that contains copies of the PIN parameters.

Return Values

The value returned by this function is one of the following.

Return Value	Description
ACE_SUCCESS	Handle to the authentication data was found and the PIN value was provided by the function.
ACE_ERR_INVALID_HANDLE	Invalid handle, could not locate the data associated with this handle.
ACE_TOO_MANY_CALLERS	An attempt was made to use a handle while a call was already in progress. This happens if a multithreaded application uses the same handle from multiple threads at the same time. This return code is not valid in this version.
ACE_INVALID_ARG	Input parameter value is not valid.

Error Handling

A value of ACE_ERR_INVALID_HANDLE is returned if the handle to the authentication data cannot be found.

Requirements

Minimum supported Authentication Manager Server Version	6.1 or later
Category	Synchronous API
Header	acexport.h
Static Library	UNIX: libaceclnt.a
Shared Library	Windows: • aceclnt.dll • aceclnt_tcp.dll • sdmsg.dll • xerces-c_3_1_vc80.dll UNIX: • libaceclnt.so • libaceclnt_tcp.so • libxerces-c-3.1.so AIX: • libaceclnt.so **Note:** Rename to libaceclnt.a. For more information, see "Using Dynamic Libraries or Static Libraries in UNIX" on page 38. • libaceclnt_tcp.so • libxerces-c-3.1.a

AceGetRealmID

Description

```
int WINAPI AceGetRealmID (
  SDI_HANDLE hdl,
  char* pRealmID )
```

The AceGetRealmID function returns the realm ID of Authentication Manager, which is a base64-encoded number that identifies the realm.

AceGetRealmID is part of the Generic Credentials API. For more information, see "Support for EAP 32 and the Generic Credential API" on page 13.

Parameters

	Parameter	Description
[in]	hdl	Handle to the current authentication session, which SDInit created.
[out]	pRealmID	Pointer to a string buffer that is ready to receive the realm ID of the Authentication Manager. The string buffer must be of the appropriate size. Realm ID is a Base64 encoded representation of the realm ID of Authentication Manager, and a null terminator. **Note:** The minimum length of the realmID must be 25 characters.

Return Values

Return Value	Description
ACE_SUCCESS	The function has successfully retrieved the authentication attribute.
ACE_ERR_INVALID_HANDLE	The handle passed was invalid.
ACE_TOO_MANY_CALLERS	An attempt was made to use a handle while a call was already in progress. This happens if a multithreaded application uses the same handle from multiple threads at the same time. This return code is not valid in this version.
ACE_INVALID_ARG	Input parameter pRealmID is not valid.

Important: There is a design constraint on the use of SDI_HANDLE. An asynchronous application of the API must avoid concurrent use of the same SDI_HANDLE. Multiple threads using the same SDI_HANDLE return incorrect results.

Error Handling

To handle errors appropriately, use the value returned by this function at a decision point in your code. A successful return allows processing to continue.

Requirements

Minimum supported Authentication Manager Server Version	6.1 or later
Category	Synchronous API
Header	acexport.h
Static Library	UNIX: libaceclnt.a
Shared Library	Windows: • aceclnt.dll • aceclnt_tcp.dll • sdmsg.dll • xerces-c_3_1_vc80.dll UNIX: • libaceclnt.so • libaceclnt_tcp.so • libxerces-c-3.1.so AIX: • libaceclnt.so **Note:** Rename to libaceclnt.a. For more information, see "Using Dynamic Libraries or Static Libraries in UNIX" on page 38. • libaceclnt_tcp.so • libxerces-c-3.1.a

AceGetShell

Description

```
int WINAPI AceGetShell(
  SDI_HANDLE hdl,
  char *shell)
```

The AceGetShell function gets a copy of the string value containing the user's shell. This value can be retrieved at any time after a successful call to AceCheck or AceClientCheck because that is when the value is retrieved from RSA Authentication Manager.

Architecture

This function is synchronous and the caller must supply, as the second argument, a pointer to a character array into which the value is copied.

Parameters

	Parameter	Description
[in]	hdl	The value of a handle originally assigned by a call to AceInit.
[out]	shell	A pointer to developer-supplied storage that receives a copy of the user data. **Note:** The minimum length of the shell field must be 65 characters.

Return Values

The function returns ACE_SUCCESS if the handle to the authentication data was found and the data value was filled in by the function.

Return Values

The value returned by this function is one of the following.

Return Value	Description
ACE_SUCCESS	The handle to the authentication data was found and the data value was filled in by the function.
ACE_ERR_INVALID_HANDLE	Invalid handle, could not locate the data associated with this handle.

ACE_TOO_MANY_CALLERS	An attempt was made to use a handle while a call was already in progress. This happens if a multithreaded application uses the same handle from multiple threads at the same time. This return code is not valid in this version.
ACE_INVALID_ARG	Input parameter shell is not valid.

Error Handling

A value of ACE_ERR_INVALID_HANDLE is returned if the handle to the authentication data cannot be found.

Requirements

Minimum supported Authentication Manager Server Version	6.1 or later
Category	Synchronous API
Header	acexport.h
Static Library	UNIX: libaceclnt.a
Shared Library	Windows: • aceclnt.dll • aceclnt_tcp.dll • sdmsg.dll • xerces-c_3_1_vc80.dll UNIX: • libaceclnt.so • libaceclnt_tcp.so • libxerces-c-3.1.so AIX: • libaceclnt.so **Note:** Rename to libaceclnt.a. For more information, see “Using Dynamic Libraries or Static Libraries in UNIX” on page 38. • libaceclnt_tcp.so • libxerces-c-3.1.a

AceGetSystemPin

Description

```
int WINAPI AceGetSystemPin(
  SDI_HANDLE hdl,
  char *val)
```

The AceGetSystemPin function gets a copy of a PIN generated by the RSA Authentication Manager for the authentication request associated with SdiHandle. The retrieval of this value is necessary if the result code from AceCheck is ACM_NEW_PIN_REQUIRED, and a call to AceGetPinParams or AceGetUserSelectable indicates that a system-generated PIN is required by the RSA Authentication Manager Administrator.

Note: To obtain all the PIN-related parameters at once, use AceGetPinParams.

Architecture

The caller of this synchronous function must supply, as the second argument, a pointer to a character string into which the PIN string is copied. This function does not allocate storage for this data on its own.

Parameters

	Parameter	Description
[in]	hdl	The value of a handle originally assigned by a call to AceInit.
[out]	PIN	A pointer to a character string that receives a copy of the PIN. **Note:** The minimum length of the PIN field must be 17 characters.

Return Values

Return Value	Description
ACE_SUCCESS	The handle to the authentication data was found and the PIN value was filled in by the function.
ACE_ERR_INVALID_HANDLE	Invalid handle, could not locate the data associated with this handle.

ACE_TOO_MANY_CALLERS	An attempt was made to use a handle while a call was already in progress. This happens if a multithreaded application uses the same handle from multiple threads at the same time. This return code is not valid in this version.
ACE_INVALID_ARG	Input parameter PIN is not valid, or this API is invalid for EAP 32. Use GetAuthAttr(ENC_SYSTEM_PIN)

Error Handling

A value of ACE_ERR_INVALID_HANDLE is returned if the handle to the authentication data cannot be found.

Requirements

Minimum supported Authentication Manager Server Version	6.1 or later
Category	Synchronous API
Header	acexport.h
Static Library	UNIX: libaceclnt.a
Shared Library	Windows: • aceclnt.dll • aceclnt_tcp.dll • sdmsg.dll • xerces-c_3_1_vc80.dll UNIX: • libaceclnt.so • libaceclnt_tcp.so • libxerces-c-3.1.so AIX: • libaceclnt.so **Note:** Rename to libaceclnt.a. For more information, see “Using Dynamic Libraries or Static Libraries in UNIX” on page 38. • libaceclnt_tcp.so • libxerces-c-3.1.a

AceGetTime

Description

```
int WINAPI AceGetTime(
  SDI_HANDLE hdl,
  UINT32BIT *val)
```

The AceGetTime function gets a copy of the current time plus the time delta between the agent and the RSA Authentication Manager. The retrieval of this value can be performed at any time after a successful call to AceInit. The value is reported in seconds.

Architecture

This function is synchronous and the caller must supply, as the second argument, a pointer to a 4-byte storage area into which the time value is copied.

Parameters

	Parameter	Description
[in]	hdl	The value of a handle originally assigned by a call to AceInit.
[out]	val	A 32-bit unsigned integer pointer to developer-supplied storage that receives a copy of the user data.

Return Values

The value returned by this function is one of the following.

Return Value	Description
ACE_SUCCESS	The handle to the authentication data was found and the data value was filled in by the function.
ACE_ERR_INVALID_HANDLE	Invalid handle, could not locate the data associated with this handle.
ACE_TOO_MANY_CALLERS	An attempt was made to use a handle while a call was already in progress. This happens if a multithreaded application uses the same handle from multiple threads at the same time. This return code is not valid in this version.
ACE_INVALID_ARG	Input parameter val is not valid.

Error Handling

A value of ACE_ERR_INVALID_HANDLE is returned if the handle to the authentication data cannot be found.

Requirements

Minimum supported Authentication Manager Server Version	6.1 or later
Category	Synchronous API
Header	acexport.h
Static Library	UNIX: libaceclnt.a
Shared Library	Windows: • aceclnt.dll • aceclnt_tcp.dll • sdmsg.dll • xerces-c_3_1_vc80.dll UNIX: • libaceclnt.so • libaceclnt_tcp.so • libxerces-c-3.1.so AIX: • libaceclnt.so **Note:** Rename to libaceclnt.a. For more information, see "Using Dynamic Libraries or Static Libraries in UNIX" on page 38. • libaceclnt_tcp.so • libxerces-c-3.1.a

AceGetUserData

Description

```
int WINAPI AceGetUserData(
  SDI_HANDLE hdl,
  unsigned int *val)
```

The AceGetUserData function gets a copy of the user data previously set as the value of the second argument in a call to AceInit or AceSetUserData. This value can be retrieved at any time. The RSA Authentication Manager does not process data stored with this function.

The purpose of the calls AceSetUserData and AceGetUserData is to supply agent developers with routines for storing and retrieving data related to the authentication in progress. Although this is most useful in cases of asynchronous operation, it is available to both synchronous and asynchronous calls.

An example of a use for these calls is in an application that operates within a server. The application might have to maintain information about the user authentication. In that case, the code must allocate some memory and pass a pointer to the memory as the 32-bit data reference to AceSetUserData. When this data is needed at a later point in the authentication process, the code can maintain only the SDI_HANDLE value. To get access to the data, the code would then call AceGetUserData to retrieve the pointer to the original data and process it. The caller of these functions is responsible for the disposition of any resources associated with the data referenced by the 32-bit value. The Authentication API furnishes the value on request.

Architecture

This function is synchronous and the caller must supply, as the second argument, a pointer to a 32-bit storage area (that is, an unsigned int) into which to copy the user data value.

Parameters

	Parameter	Description
[in]	hdl	The value of a handle originally assigned by a call to AceInit.
[out]	val	A pointer to a 32-bit storage area (an unsigned integer) that receives a copy of the user data.

Return Values

Return Value	Description
ACE_SUCCESS	The handle to the authentication data was found and the data value was filled in by the function.

ACE_ERR_INVALID_HANDLE	Invalid handle, could not locate the data associated with this handle.
ACE_TOO_MANY_CALLERS	An attempt was made to use a handle while a call was already in progress. This happens if a multithreaded application uses the same handle from multiple threads at the same time. This return code is not valid in this version.
ACE_INVALID_ARG	Input parameter value is not valid.

Error Handling

A value of ACE_ERR_INVALID_HANDLE is returned if the handle to the authentication data cannot be found.

Requirements

Minimum supported Authentication Manager Server Version	6.1 or later
Category	Synchronous API
Header	acexport.h
Static Library	UNIX: libaceclnt.a
Shared Library	Windows: • aceclnt.dll • aceclnt_tcp.dll • sdmsg.dll • xerces-c_3_1_vc80.dll UNIX: • libaceclnt.so • libaceclnt_tcp.so • libxerces-c-3.1.so AIX: • libaceclnt.so **Note:** Rename to libaceclnt.a. For more information, see "Using Dynamic Libraries or Static Libraries in UNIX" on page 38. • libaceclnt_tcp.so • libxerces-c-3.1.a

AceGetUserSelectable

Description

```
int WINAPI AceGetUserSelectable(
  SDI_HANDLE hdl,
  char *val)
```

The AceGetUserSelectable function determines whether the user associated with the authentication request has the ability to select a new PIN value. The retrieval of this value might be necessary when a value of ACM_NEW_PIN_REQUIRED is returned by AceCheck.

Note: To obtain all the PIN-related parameters at once, use AceGetPinParams.

Architecture

This synchronous function fills in the second argument passed to it with a value that indicates whether the user can select a new PIN.

Parameters

	Parameter	Description
[in]	hdl	The value of a handle originally assigned by a call to AceInit.
[out]	val	A pointer to a character that is assigned the user-selectable value.

Return Values

Return Value	Description
ACE_SUCCESS	The handle to the authentication data was found and the user-selectable value was filled in by the function.
ACE_ERR_INVALID_HANDLE	Invalid handle, could not locate the data associated with this handle.
ACE_TOO_MANY_CALLERS	An attempt was made to use a handle while a call was already in progress. This happens if a multithreaded application uses the same handle from multiple threads at the same time. This return code is not valid in this version.
ACE_INVALID_ARG	Input parameter val is not valid.

Val data	Description
CANNOT_CHOOSE_PIN	The user must either accept a system-generated PIN or cancel the operation and leave the token in New PIN mode.
MUST_CHOOSE_PIN	The user must create his or her own PIN. The user is not given the option of receiving a system-generated PIN.
USER_SELECTABLE	The user can either create a PIN or request a system-generated one. For Authentication Manager 8.0 or later, this value is not supported.

Error Handling

A value of ACE_ERR_INVALID_HANDLE is returned if the handle to the authentication data cannot be found.

Requirements

Minimum supported Authentication Manager Server Version	6.1 or later
Category	Synchronous API
Header	acexport.h
Static Library	UNIX: libaceclnt.a
Shared Library	Windows: • aceclnt.dll • aceclnt_tcp.dll • sdmsg.dll • xerces-c_3_1_vc80.dll UNIX: • libaceclnt.so • libaceclnt_tcp.so • libxerces-c-3.1.so AIX: • libaceclnt.so **Note:** Rename to libaceclnt.a. For more information, see "Using Dynamic Libraries or Static Libraries in UNIX" on page 38. • libaceclnt_tcp.so • libxerces-c-3.1.a

AceInit

Description

```
int WINAPI AceInit(
  LP_SDI_HANDLE pSdiHandle,
  unsigned int userData,
  void (WINAPI*appCallback)(SDI_HANDLE));
```

AceInit initializes a socket and makes a call to the RSA Authentication Manager to verify communication. AceInit must be called first in the sequence of calls for each authentication request.

The value of the argument userData can be retrieved with AceGetUserData. For more information on the function and uses of userData, see “AceGetUserData”on page 104.

Architecture

This asynchronous function returns immediately with a status value. If the status is ACE_PROCESSING, other threads complete the initialization process.

The AceInit function allocates storage for a data structure that is to be used for a particular authentication process, sets variables within that structure, enables other threads, and returns a value to the caller. When the threads invoked by AceInit finish, the developer-defined function is called (if one has been supplied).

The developer passes a pointer to a handle and a function pointer to AceInit, which assigns a unique value to the handle. In subsequent API calls, this handle identifies the particular authentication request. The developer can supply a callback function with the following prototype:

```
void WINAPI Callback(SDI_HANDLE);
```

The callback function is called when AceInit finishes.

Parameters

	Parameter	Description
[out]	pSdiHandle	A pointer to a handle that is assigned by the AceInit function.
[in]	userData	A 32-bit value that holds data to be associated with the authentication request. It can be passed a handle or any other pointer type.
[in]	appCallback	A pointer to a function written by the developer. This function returns nothing. The handle to the data area specific to the authentication process is passed to this function. A NULL value can be passed to AceInit instead of a valid function pointer.

Return Values

Return Value	Description
ACE_PROCESSING	Normal return.
ACE_CFGFILE_NOT_FOUND	The **sdconf.rec** file could not be opened.
ACE_CFGFILE_READ_FAIL	Unable to read **sdconf.rec** file.

Error Handling

To handle errors appropriately, use the value returned by this function at a decision point in your code. A successful return allows processing to continue. If ACE_PROCESSING is returned, your code must eventually call AceClose.

Checking the Results

If the caller supplied a callback function, it can verify the result of the authentication operation by calling the AceGetAuthenticationStatus function.

Result	Description
ACM_OK	The connection with RSA Authentication Manager was successful. Use AceCheck or AceClientCheck to proceed with the authentication.
ACM_NO_SERVER	The connection with RSA Authentication Manager failed. End the authentication attempt with AceClose.

Requirements

Minimum supported Authentication Manager Server Version	6.1 or later
Category	Asynchronous API
Header	acexport.h
Static Library	UNIX: libaceclnt.a
Shared Library	Windows: • aceclnt.dll • aceclnt_tcp.dll • sdmsg.dll • xerces-c_3_1_vc80.dll UNIX: • libaceclnt.so • libaceclnt_tcp.so • libxerces-c-3.1.so AIX: • libaceclnt.so **Note:** Rename to libaceclnt.a. For more information, see “Using Dynamic Libraries or Static Libraries in UNIX” on page 38. • libaceclnt_tcp.so • libxerces-c-3.1.a

AceInitEx

Description

```
int WINAPI AceInitEx(
        LP_SDI_HANDLE pSdiHandle,
        unsigned int userData,
        void (WINAPI*appCallback)(SDI_HANDLE),
        RSA_DA_AGENT_TYPE agentType) (This flag is only supported for backward
compatibility);
```

It performs the same initialization tasks as AceInit and is supported only for backward compatibility. The call is supported, but the offline authentication feature is not.

Architecture

This asynchronous function returns immediately with a status value. If the status is ACE_PROCESSING, other threads complete the initialization process. The AceInitEx function allocates storage for a data structure that is to be used for a particular authentication process, sets variables within that structure, enables other threads, and returns a value to the caller. When the threads invoked by AceInitEx finish, the developer-defined function is called (if one has been supplied). The developer passes a pointer to a handle and a function pointer to AceInitEx, which assigns a unique value to the handle. In subsequent API calls, this handle identifies the particular authentication request. The developer can supply a callback function with the following prototype:

```
void WINAPI Callback(SDI_HANDLE);
```

The callback function is called when AceInitEx finishes.

Parameters

	Parameter	Description
[out]	pSdiHandle	A pointer to a handle that is assigned by the **AceInit** function.
[in]	userData	A 32-bit value that holds data to be associated with the authentication request. It can be passed a handle or any other pointer type.
[in]	appCallback	A pointer to a function written by the developer. This function returns nothing. The handle to the data area specific to the authentication process is passed to this function. A NULL value can be passed to AceInit instead of a valid function pointer.
[in]	agentType	This flag is related to the offline authentication feature. It is no longer supported in this version and will be ignored.

Return Values

The value returned by the AceInit function is one of the following.

Return Value	Description
ACE_PROCESSING	Normal return.
ACE_CFGFILE_NOT_FOUND	The **sdconf.rec** file could not be opened.
ACE_CFGFILE_READ_FAIL	Unable to read **sdconf.rec** file.

Error Handling

To handle errors appropriately, use the value returned by this function at a decision point in your code. A successful return allows processing to continue. If ACE_PROCESSING is returned, your code must eventually call AceClose.

RSA_DA_AGENT_TYPE

This tells the agent what type of agent is installed; this is no longer supported in this version.

Checking the Results

If the caller supplied a callback function, it can verify the result of the authentication operation by calling the AceGetAuthenticationStatus function. The status code could be one of the following values.

Result	Description
ACM_OK	The connection with RSA Authentication Manager was successful. Use AceCheck or AceClientCheck to proceed with the authentication.
ACM_NO_SERVER	The connection with RSA Authentication Manager failed. End the authentication attempt with AceClose.

Requirements

Minimum supported Authentication Manager Server Version	6.1 or later
Category	Asynchronous API
Header	acexport.h
Static Library	UNIX: libaceclnt.a
Shared Library	Windows: • aceclnt.dll • aceclnt_tcp.dll • sdmsg.dll • xerces-c_3_1_vc80.dll UNIX: • libaceclnt.so • libaceclnt_tcp.so • libxerces-c-3.1.so AIX: • libaceclnt.so **Note:** Rename to libaceclnt.a. For more information, see “Using Dynamic Libraries or Static Libraries in UNIX” on page 38. • libaceclnt_tcp.so • libxerces-c-3.1.a

AceInitialize

Description

```
SD_BOOL WINAPI AceInitialize( void );
```

Note: To use configuration files (**rsa_api.properties** and **sdconf.rec**), put these files in the **aceclnt.dll** location (VAR_ACE on UNIX or AIX environments).

AceInitialize initializes the worker threads and loads the mandatory **sdconf.rec** file, and optionally, the **sdopts.rec** file, into memory. The **sdconf.rec** file is the configuration file created during installation of the RSA Authentication Manager software. The **sdopts.rec** file is created manually to effect manual load balancing.

In the Windows environment, the API expects to find the configuration files in the ***%SystemRoot%*\system32** directory if the configuration file is not found in the **aceclnt.dll** location. In the UNIX environment, the API expects to find the configuration files in the **var/ace** directory (or the directory specified by the $VAR_ACE system variable). Make certain that the configuration files are present in the directory that corresponds to your environment.

Do no edit the config.xml, bootstrap.xml, and root.cer files. If configuration files were modified by mistake, then those files should be deleted so that new configuration files can be downloaded from the server. When the configuration files are corrupted, AceInitialize does not update the configuration data.

Applications running with elevated privileges that let the environment variable $VAR_ACE control the path to the configuration files are subject to path manipulation. Such applications must always explicitly specify all paths using AceInitializeEx API, rather than relying on the environment.

AceInitialize must be called before any other API function is called, and must be called only once. The AceInitialize function replaces the creadcfg call (which is no longer supported). You must replace any creadcfg calls in older custom agents with calls to AceInitialize.

Important: It is required that your code always calls AceInitialize once, before calling any other API function, in both Windows and UNIX environments.

Although it is the developer's responsibility to call AceInitialize, you can avoid potential error conditions by using AceInit, AceStartAuth, SD_Init, or sd_init. Any of these functions detects whether AceInitialize has already been called, and calls it, if necessary. However, if your application calls AceInit, AceStartAuth, or SD_Init in a multithreaded environment, and multiple automatic calls to AceInitialize occur at the same time, those functions return an error. In this case, your code must make a call to AceInitialize once before calling AceInit, AceStartAuth, or SD_Init.

Architecture

The AceInitialize function creates the internal structures and the worker threads that operate during authentication requests. If the function is called more than once, it returns success. If more than two threads call this function for the first time simultaneously, one of the calls returns SD_FALSE. This happens if your code does not call AceInitialize directly and instead relies on the automatic call from AceInit, AceStartAuth, SD_Init, or sd_init.

Parameters

None.

Return Values

If the AceInitialize function initializes the internal state of the library successfully, it returns SD_TRUE. If there is any failure, it returns SD_FALSE.

Requirements

Minimum supported Authentication Manager Server Version	6.1 or later
Category	Synchronous API
Header	acexport.h
Static Library	UNIX: libaceclnt.a
Shared Library	Windows: • aceclnt.dll • aceclnt_tcp.dll • sdmsg.dll • xerces-c_3_1_vc80.dll UNIX: • libaceclnt.so • libaceclnt_tcp.so • libxerces-c-3.1.so AIX: • libaceclnt.so **Note:** Rename to libaceclnt.a. For more information, see "Using Dynamic Libraries or Static Libraries in UNIX" on page 38. • libaceclnt_tcp.so • libxerces-c-3.1.a

AceInitializeEx

Description

```
SD_BOOL WINAPI AceInitializeEx(
SD_CHAR *configFile,
SD_CHAR *reserved1,
SD_U32 reserved2);
```

AceInitializeEx has the same functionality as AceInitialize except that AceInitializeEx allows you to specify the path to the configuration files (**rsa_api.properties** and **sdconf.rec**).

Like AceInitialize, AceInitializeEx initializes the worker threads and loads the mandatory **sdconf.rec** file, and optionally, the **sdopts.rec** file, into memory. The **sdconf.rec** file is the configuration file created during installation of the RSA Authentication Manager software. The **sdopts.rec** file is created manually to affect manual load balancing.

Make certain that the configuration files are present in the directory that you specify.

You must call AceInitializeEx before calling any other API function, and you must call it only once.

Important: It is required that your code always calls AceInitializeEx once, before calling any other API function, in both Windows and UNIX environments.

Although it is the developer's responsibility to call AceInitializeEx/AceInitialize, you can avoid potential error conditions by using AceInit, AceStartAuth, or SD_Init. Any of these functions detects whether AceInitializeEx/AceInitialize has already been called and calls AceInitialize, if necessary. However, if your application calls AceInit, AceStartAuth, or SD_Init in a multithreaded environment, and multiple automatic calls to AceInitializeEx occur simultaneously, those functions return an error. In this case, your code must make a call to AceInitializeEx once before calling AceInit, AceStartAuth, or SD_Init.

Architecture

The AceInitializeEx function creates the internal structures and the worker threads that operate during authentication requests. If the function is called more than once, it returns success. If more than two threads call this function for the first time simultaneously, one of the calls returns SD_FALSE. This can happen if your code does not call AceInitializeEx directly and instead relies on the automatic call from AceInit, AceStartAuth, SD_Init, or sd_init.

Parameters

	Parameter	Description
[in]	configFile	Possible values are described below. **Note:** The maximum length depends on the maximum file path length supported by specific OS. This length includes the length of the default configuration directory, which will be created by SDK (unless the paths are explicitly specified in rsa_api.properties).
	reserved1	Reserved for internal use. Must be set to NULL.
	reserved2	Reserved for internal use. Must be set to 0.

The configFile can accept any of the below configuration file paths:

- Full path to sdconf.rec file:

  To use rsa_api.properties in this case, it can be placed in the same directory as sdconf.rec. If it is present, the properties other than SDCONF_LOC will be parsed from the rsa_api.properties file.

- Full path to rsa_api.properties file:

  All the properties will be parsed from the rsa_api.properties file which is passed as argument. If the properties file doe not specify SDCONF_LOC, then it should be located in the same directory as rsa_api.properties.

- Full path to directory in which any of the above file is located:

  If the rsa_api.properties file is present in the specified directory, all the properties will be parsed from it. If the sdconf.rec file is not found in the specified directory, then the API uses the location specified by the SDCONF_LOC property in rsa_api.properties.

  If rsa_api.properties is not present in the directory, then sdconf.rec should be placed in it.

Return Values

If the **AceInitializeEx** function initializes the internal state of the library successfully, it returns **SD_TRUE**. If there is any failure, it returns **SD_FALSE**.

Requirements

Minimum supported Authentication Manager Server Version	6.1 or later
Category	Synchronous API
Header	acexport.h
Static Library	UNIX: libaceclnt.a
Shared Library	Windows: • aceclnt.dll • aceclnt_tcp.dll • sdmsg.dll • xerces-c_3_1_vc80.dll UNIX: • libaceclnt.so • libaceclnt_tcp.so • libxerces-c-3.1.so AIX: • libaceclnt.so **Note:** Rename to libaceclnt.a. For more information, see "Using Dynamic Libraries or Static Libraries in UNIX" on page 38. • libaceclnt_tcp.so • libxerces-c-3.1.a

AceLock

Note: This function is not supported in this version of the API .

Description

```
int WINAPI AceLock(
  SDI_HANDLE SdiHandle,
  void (WINAPI*appCallback)(SDI_HANDLE));
```

The AceLock function verifies that a non-empty user name has been set by a previous call to AceSetUsername, and then sends a lock request to the RSA Authentication Manager as part of the two-step authentication process. For more information, see "Load Balancing" on page 14.

The AceLock function must be called only after successful calls to AceInit and AceSetUsername for the specific authentication request.

Architecture

This asynchronous function returns immediately with a status value. If the status is ACE_PROCESSING, other threads perform the rest of the task.

The AceLock function attempts to find the data associated with this specific authentication process through the unique handle value. If the data is found, AceLock enables other threads to process the data. When the threads invoked by AceLock finish, the developer-defined function is called (if one has been supplied).

Parameters

	Parameter	Description
[in]	SdiHandle	The value of a handle originally assigned by a call to AceInit.
[in]	appCallback	A pointer to a function written by the developer. This function returns nothing. The handle to the data area specific to the authentication process is passed to this function. A NULL value can be passed instead of a valid function pointer.

Return Values

Return Value	Description
ACE_PROCESSING	Normal return.
ACE_ERR_INVALID_HANDLE	AceLock could not locate the data associated with the handle.
ACE_UNDEFINED_USERNAME	The user name has yet to be set for the authentication request.
ACE_TOO_MANY_CALLERS	An attempt was made to use a handle while a call was already in progress. This happens if a multithreaded application uses the same handle from multiple threads at the same time. This return code is not valid in this version.

Error Handling

To handle errors appropriately, use the value returned by this function at a decision point in your code. A successful return allows processing to continue.

Checking the Results

If the caller supplied a callback function, it can verify the result of the authentication operation by calling the AceGetAuthenticationStatus function. The status code is one of the following values.

Result	Description
ACM_OK	The user name lock request has been sent to the RSA Authentication Manager. Use the AceCheck function to process the passcode.
ACM_ACCESS_DENIED	Communication with the RSA Authentication Manager has failed. The failure might result either due to a communication time-out or because there is a mismatch in the node secret between the agent and the server. Work with the RSA Authentication Manager administrator to troubleshoot the problem.

Requirements

Minimum supported Authentication Manager Server Version	6.1
Category	Asynchronous API
Header	acexport.h
Static Library	UNIX: libaceclnt.a
Shared Library	Windows: • aceclnt.dll • aceclnt_tcp.dll • sdmsg.dll • xerces-c_3_1_vc80.dll UNIX: • libaceclnt.so • libaceclnt_tcp.so • libxerces-c-3.1.so AIX: • libaceclnt.so **Note:** Rename to libaceclnt.a. For more information, see “Using Dynamic Libraries or Static Libraries in UNIX” on page 38. • libaceclnt_tcp.so • libxerces-c-3.1.a

AceRefreshIP

Note: This function is not supported in this release of the API and will always return success.

Description

```
void AceRefreshIP()
```

This synchronous function refreshes the client IP on each authentication request. This is useful when the agent's IP is not static. However, it requires auto-registration facility to be enabled at the Authentication Manager server.

Note: Never use it if this instance of agent allows more than one authentication at the same time.

Requirements

Minimum supported Authentication Manager Server Version	6.1 or later
Category	Synchronous API
Header	acexport.h
Static Library	UNIX: libaceclnt.a
Shared Library	Windows: • aceclnt.dll • aceclnt_tcp.dll • sdmsg.dll • xerces-c_3_1_vc80.dll UNIX: • libaceclnt.so • libaceclnt_tcp.so • libxerces-c-3.1.so AIX: • libaceclnt.so **Note:** Rename to libaceclnt.a. For more information, see “Using Dynamic Libraries or Static Libraries in UNIX” on page 38. • libaceclnt_tcp.so • libxerces-c-3.1.a

AceSendNextPasscode

Description

```
int WINAPI AceSendNextPasscode(
  SDI_HANDLE SdiHandle,
  void (WINAPI*appCallback)(SDI_HANDLE));
```

The AceSendNextPasscode function takes a successive credential and checks its validity. The next passcode value is set by a previous call to AceSetNextPasscode before this function is called. Because AceSendNextPasscode does not display the Next Code prompt, the integrating application must perform all I/O.

Call the AceSendNextPasscode function if the result code from AceCheck is ACM_NEXT_CODE_REQUIRED. Note that an ACM_NEXT_CODE_REQUIRED result code can only occur after AceCheck has finished. Therefore, the developer-defined callback can check for this return value from AceCheck using AceGetAuthenticationStatus. The immediate return value from AceCheck is never set to this value.

Architecture

The AceSendNextPasscode function is an asynchronous function that returns immediately with a status value. If the status is ACE_PROCESSING, other threads perform the rest of the task.

The AceSendNextPasscode function attempts to find the data associated with the specific authentication process through the unique handle value. If the data is found, AceSendNextPasscode enables another thread to process the data. When the threads invoked by AceSendNextPasscode finish, the developer-defined function is called (if one is included).

Parameters

	Parameter	Description
[in]	SdiHandle	The value of a handle originally assigned by a call to AceInit.
[in]	appCallback	This function returns nothing. The handle to the data area specific to the authentication process is passed to this function. A NULL value can be passed instead of a valid function pointer.

Return Values

Return Value	Description
ACE_PROCESSING	Normal return.
ACE_ERR_INVALID_HANDLE	AceSendNextPasscode could not locate the data associated with the handle, or user has not authenticated successfully using AceCheck or SD_Check API.
ACE_UNDEFINED_NEXT_PASSCODE	The next passcode has not yet been set for the authentication request.
ACE_TOO_MANY_CALLERS	An attempt was made to use a handle while a call was already in progress. This happens if a multithreaded application uses the same handle from multiple threads at the same time. This return code is not valid in this version.

Error Handling

To handle errors appropriately, use the value returned by this function at a decision point in your code. A successful return allows processing to continue.

Checking the Results

If the caller supplied a callback function, it can verify the result of the authentication operation by calling the AceGetAuthenticationStatus function.

Result	Description
ACM_OK	The user was successfully authenticated. Use the AceGetShell function to access the shell field.
ACM_ACCESS_DENIED	The user failed authentication.

Requirements

Minimum supported Authentication Manager Server Version	6.1 or later
Category	Asynchronous API
Header	acexport.h
Static Library	UNIX: libaceclnt.a
Shared Library	Windows: • aceclnt.dll • aceclnt_tcp.dll • sdmsg.dll • xerces-c_3_1_vc80.dll UNIX: • libaceclnt.so • libaceclnt_tcp.so • libxerces-c-3.1.so AIX: • libaceclnt.so **Note:** Rename to libaceclnt.a. For more information, see “Using Dynamic Libraries or Static Libraries in UNIX” on page 38. • libaceclnt_tcp.so • libxerces-c-3.1.a

AceSendPin

Description

```
int WINAPI AceSendPin(
  SDI_HANDLE hdl,
  void (WINAPI*appCallback)(SDI_HANDLE SdiHandle));
```

The AceSendPin function transmits a new PIN to RSA Authentication Manager for storage in a token record. AceSendPin must be called only when the result code ACM_NEW_PIN_REQUIRED is set by AceCheck.

Note: Do not consider users to be authenticated as soon as they have completed the New PIN operation. Users must enter the new PIN to be authenticated.

Architecture

This asynchronous function returns immediately with a status value. If the status is ACE_PROCESSING, threads are enabled to process the data.

The AceSendPin function attempts to find the data associated with the specific authentication process through the unique handle value. If the data is found, AceSendPin calls another thread to process the data.

When the threads invoked by AceSendPin finish, the developer-defined function is called (if one has been supplied). AceSendPin allows the caller to attempt to change the PIN in the RSA Authentication Manager token record. The PIN value must have been previously set with a call to AceSetPin before calling this function.

Parameters

	Parameter	Description
[in]	hdl	The value of a handle originally assigned by a call to AceInit.
[in]	appCallback	A pointer to a function written by the developer. This function returns nothing. The handle to the data area specific to the authentication process is passed to this function. A NULL value can be passed instead of a valid function pointer.

Return Values

Return Value	Description
ACE_PROCESSING	Normal return.

ACE_ERR_INVALID_HANDLE	AceSendPin could not locate the data associated with the handle, or user has not authenticated successfully using AceCheck or SD_Check API.
ACE_UNDEFINED_PIN	The user's PIN has not yet been set for the authentication request.
ACE_TOO_MANY_CALLERS	An attempt was made to use a handle while a call was already in progress. This happens if a multithreaded application uses the same handle from multiple threads at the same time. This return code is not valid in this version.

Error Handling

To handle errors appropriately, use the value returned by this function at a decision point in your code. A successful return allows processing to continue.

Checking the Results

If the caller supplied a callback function, it can verify the result of the authentication operation by calling the AceGetAuthenticationStatus function. The status code is one of the following values.

Result	Description
ACM_NEW_PIN_ACCEPTED	The RSA Authentication Manager has accepted the new PIN. The user is now required to authenticate with the new PIN.
ACM_NEW_PIN_REJECTED	The RSA Authentication Manager rejected the new PIN. The PIN might not have matched the parameters set in the return from AceCheck.

Requirements

Minimum supported Authentication Manager Server Version	6.1 or later
Category	Asynchronous API
Header	acexport.h
Static Library	UNIX: libaceclnt.a
Shared Library	Windows: • aceclnt.dll • aceclnt_tcp.dll • sdmsg.dll • xerces-c_3_1_vc80.dll UNIX: • libaceclnt.so • libaceclnt_tcp.so • libxerces-c-3.1.so AIX: • libaceclnt.so **Note:** Rename to libaceclnt.a. For more information, see “Using Dynamic Libraries or Static Libraries in UNIX” on page 38. • libaceclnt_tcp.so • libxerces-c-3.1.a

AceSetAuthAttr

Description

```
int WINAPI AceSetAuthAttr (
   SDI_HANDLE SdiHandle,
   RSA_AUTH_SET_ATTR authAttribute,
   void *attrValue,
   SD_U32 attrSize )
```

The AceSetAuthAttr function requests a particular authentication attribute from Authentication Manager. The definitions of the attributes and associated data are described in the header file **acexport.h**. Attributes are set before a synchronous or asynchronous authentication and are delivered with the authentication request to the server.

AceSetAuthAtt**r** can be used as part of the Generic Credentials API. For more information, see "Support for EAP 32 and the Generic Credential API" on page 13.

Parameters

	Parameter	Description
[in]	SdiHandle	Handle to the current authentication session, which SDInit created.
[in]	authAttribute	The identifier of the requested authentication attribute.
[in]	attrValue	The raw authentication attribute data.
[in]	attrSize	The size of the authentication attribute data.

Return Values

The value returned by this function is one of the following.

Return Value	Description
ACE_SUCCESS	The call was successful.
ACE_INVALID_ARG	An invalid argument was supplied to an API function, or the attrValue is not large enough to store the attribute data.
ACE_ERR_INVALID_HANDLE	Invalid handle, could not locate the data associated with this handle.
ACE_TOO_MANY_CALLERS	An attempt was made to use a handle while a call was already in progress. This happens if a multithreaded application uses the same handle from multiple threads at the same time. This return code is not valid in this version.

Important: There is a design constraint on the use of SDI_HANDLE. An asynchronous application of the API must avoid concurrent use of the same SDI_HANDLE. Multiple threads using the same SDI_HANDLE return incorrect results.

Error Handling

To handle errors appropriately, use the value returned by this function at a decision point in your code. A successful return allows processing to continue.

RSA_AUTH_SET_ATTR

Value	Description
RSA_AUTH_SET_ATTR_RADIUS_PROFILE	Third-party Auth Extension
RSA_AUTH_SET_ATTR_RESERVED	Reserved value
RSA_AUTH_SET_ATTR_RESERVED2	Reserved value
RSA_AUTH_SET_ATTR_DOMAIN_CLIENT_VERSION	Indicates if the domain agent client is 6.1 or later
RSA_AUTH_SET_ATTR_AGENT_TYPE	Indicates if it is a LAC or DAH installation
RSA_AUTH_SET_ATTR_RADIUS_EXTENSIONS	Radius User Extensions requested
RSA_AUTH_SET_ATTR_EAP32	Indicates EAP32 authentication extension is enabled

Requirements

Minimum supported Authentication Manager Server Version	6.1 or later
Category	Synchronous API
Header	acexport.h
Static Library	UNIX: libaceclnt.a
Shared Library	Windows: • aceclnt.dll • aceclnt_tcp.dll • sdmsg.dll • xerces-c_3_1_vc80.dll UNIX: • libaceclnt.so • libaceclnt_tcp.so • libxerces-c-3.1.so AIX: • libaceclnt.so **Note:** Rename to libaceclnt.a. For more information, see "Using Dynamic Libraries or Static Libraries in UNIX" on page 38. • libaceclnt_tcp.so • libxerces-c-3.1.a

AceSetCredential

Description

```
int WINAPI AceSetCredential (
  SDI_HANDLE hdl,
  RSA_AUTH_CRED_TYPE credType,
  void *credBuf,
  SD_U32 credLen )
```

The AceSetCredential function sets the generic credential to authenticate with either AceCheck or AceCheckClient. It is called after a successful return from AceInit and before a call to AceCheck. AceCheck returns with an error if the passcode value has not yet been set.The generic credential is either the passcode or the EAP 32 binary credential.

AceSetCredential is part of the Generic Credentials API. For more information, see "Support for EAP 32 and the Generic Credential API" on page 13.

Architecture

The caller of this function must supply as the second argument a pointer to the character string containing the passcode value to be copied.

Parameters

	Parameter	Description
[in]	hdl	Handle to the current authentication session, which SDInit created.
[in]	credType	The generic credential type.
[in]	credBuf	The raw credential data.
[in]	credLen	The size of the raw credential data.

Return Values

Return Value	Description
ACE_SUCCESS	The call was successful.
ACE_ERR_INVALID_HANDLE	The handle to the authentication data is incorrect.
ACE_TOO_MANY_CALLERS	An attempt was made to use a handle while a call was already in progress. This happens if a multithreaded application uses the same handle from multiple threads at the same time. This return code is not valid in this version.
ACE_INVALID_ARG	The format of the credential is invalid.

Important: There is a design constraint on the use of SDI_HANDLE. An asynchronous application of the API must avoid concurrent use of the same SDI_HANDLE. Multiple threads using the same SDI_HANDLE return incorrect results.

Error Handling

To handle errors appropriately, use the value returned by this function at a decision point in your code. A successful return allows processing to continue.

RSA_AUTH_CRED_TYPE

Value	Description
RSA_AUTH_CRED_TYPE_PASSCODE	Pass-code: Pin, Token-code, or both
RSA_AUTH_CRED_TYPE_EAP32	EAP32 Authentication Credential
RSA_AUTH_CRED_TYPE_EAP32_RESUME	EAP32 Session Resumption Auth Credential

Requirements

Minimum supported Authentication Manager Server Version	6.1 or later
Category	Synchronous API
Header	acexport.h
Static Library	UNIX: libaceclnt.a
Shared Library	Windows: • aceclnt.dll • aceclnt_tcp.dll • sdmsg.dll • xerces-c_3_1_vc80.dll UNIX: • libaceclnt.so • libaceclnt_tcp.so • libxerces-c-3.1.so AIX: • libaceclnt.so **Note:** Rename to libaceclnt.a. For more information, see “Using Dynamic Libraries or Static Libraries in UNIX” on page 38. • libaceclnt_tcp.so • libxerces-c-3.1.a

AceSetLoginPW

Note: This function is not supported in this release of the API and will always return success.

This function sends the user’s logon password to the RSA Authentication Manager for a database update. It does not require a successful authentication, but requires that the offline service is running on the local system.

Note: In this release of the API, this function is for the Windows platform only. If you need this functionality for *NIX platforms, you can set the users' passwords through the Active Directory to Authentication Manager, and then retrieve them using AceGetLoginPW() API.

Description

```
int WINAPI AceSetLoginPW (
   SDI_HANDLE          hdl,
   SD_CHAR *pchPWBuffer,
   SD_U32 u32PWLen)
```

To call AceSetLoginPW, the API must have established communication with the offline service (through a call to SD_Init, or to SD_InitEx by setting bSupportOA to SD_TRUE). In addition, you must configure RSA Authentication Manager to allow password integration.

Parameters

	Parameter	Description
[in]	hdl	The value of a handle originally assigned by a call to SD_Init or SD_InitEx.
[in]	loginPassword	A pointer to a character buffer that has been allocated by the caller which contains a copy of the password to be sent to RSA Authentication Manager.
[in]	pwLen	An unsigned 32-bit value that specifies the length of the caller-supplied password pointed to by pchPWBuffer. The maximum size of the password is 128 bytes.

Return Values

Return Value	Description
ACE_SUCCESS	The password is successfully updated.
Other errors	Indicates failure to update the password.
ACE_ERR_INVALID_HANDLE	Invalid handle, could not locate the data associated with this handle.
ACE_TOO_MANY_CALLERS	An attempt was made to use a handle while a call was already in progress. This happens if a multithreaded application uses the same handle from multiple threads at the same time. This return code is not valid in this version.
ACE_INVALID_ARG	Input parameter loginPassword or pwLen is not valid.

Requirements

Minimum supported Authentication Manager Server Version	6.1 or later
Category	Synchronous API
Header	acexport.h
Static Library	UNIX: libaceclnt.a

Minimum supported Authentication Manager Server Version	6.1 or later
Shared Library	Windows: • aceclnt.dll • aceclnt_tcp.dll • sdmsg.dll • xerces-c_3_1_vc80.dll UNIX: • libaceclnt.so • libaceclnt_tcp.so • libxerces-c-3.1.so AIX: • libaceclnt.so **Note:** Rename to libaceclnt.a. For more information, see “Using Dynamic Libraries or Static Libraries in UNIX” on page 38. • libaceclnt_tcp.so • libxerces-c-3.1.a

AceSetNextCredential

Description

```
int WINAPI AceSetNextCredential (
  SDI_HANDLE hdl,
  void *nextBuf,
  SD_U32 nextLen )
```

The AceSetNextCredential function sets the nextPasscode value for the specific authentication request associated with the value of the handle passed to it. AceSetNextCredential is called after a successful return from AceCheck. It is used when the task completed by AceCheck reveals (using a call to AceGetAuthenticationStatus) that Next Tokencode mode is currently enabled. It must be called before calling AceSendNextPasscode.

AceSetNextCredential is part of the Generic Credentials API. For more information, see "Support for EAP 32 and the Generic Credential API" on page 13.

Note: The credential type must be the same as what was used during the initial authentication request.

Parameters

	Parameter	Description
[in]	hdl	Handle to the current authentication session, which SDInit created.
[in]	nextBuf	The raw next credential value.
[in]	nextLen	The size of the next credential data.

Return Values

The value returned by this function is one of the following.

Return value	Description
ACE_SUCCESS	The call was successful.
ACE_ERR_INVALID_HANDLE	The handle to the authentication data is incorrect.
ACE_TOO_MANY_CALLERS	An attempt was made to use a handle while a call was already in progress. This happens if a multithreaded application uses the same handle from multiple threads at the same time. This return code is not valid in this version.
ACE_INVALID_ARG	The format of the credential is invalid.

ACM_ACCESS_DENIED	This API must be used with Generic credential APIs only.

Important: There is a design constraint on the use of SDI_HANDLE. An asynchronous application of the API must avoid concurrent use of the same SDI_HANDLE. Multiple threads using the same SDI_HANDLE return incorrect results.

Error Handling

To handle errors appropriately, use the value returned by this function at a decision point in your code. A successful return allows processing to continue.

Requirements

Minimum supported Authentication Manager Server Version	6.1 or later
Category	Synchronous API
Header	acexport.h
Static Library	UNIX: libaceclnt.a
Shared Library	Windows: • aceclnt.dll • aceclnt_tcp.dll • sdmsg.dll • xerces-c_3_1_vc80.dll UNIX: • libaceclnt.so • libaceclnt_tcp.so • libxerces-c-3.1.so AIX: • libaceclnt.so **Note:** Rename to libaceclnt.a. For more information, see "Using Dynamic Libraries or Static Libraries in UNIX" on page 38. • libaceclnt_tcp.so • libxerces-c-3.1.a

AceSetNextPasscode

Description

```
int WINAPI AceSetNextPasscode(
  SDI_HANDLE SdiHandle,
  char *nextPasscode)
```

The AceSetNextPasscode function sets the nextPasscode value for the specific authentication request associated with the value of the handle passed to it. This function is called after a successful return from AceCheck. It is used when the task completed by AceCheck reveals (using a call to AceGetAuthenticationStatus) that Next Tokencode mode is currently enabled. It must be called before calling AceSendNextPasscode.

Architecture

This function is synchronous.

Parameters

	Parameter	Description
[in]	hdl	The value of a handle originally assigned by a call to AceInit.
[in]	nextPasscode	A pointer to a character string containing the passcode value to be copied.

Return Values

The function returns ACE_SUCCESS if it finds the handle to the authentication data and successfully copies the next passcode value.

Return value	Description
ACE_ERR_INVALID_HANDLE	Invalid handle, could not locate the data associated with this handle.
ACE_TOO_MANY_CALLERS	An attempt was made to use a handle while a call was already in progress. This happens if a multithreaded application uses the same handle from multiple threads at the same time. This return code is not valid in this version.
ACE_INVALID_ARG	Input parameter nextPasscode is not valid.

Error Handling

A value of ACE_ERR_INVALID_HANDLE is returned if the function cannot find the handle to the authentication data. A value of ACE_INVALID_ARG is returned if nextPasscode is NULL or its length is too small or too large.

Requirements

Minimum supported Authentication Manager Server Version	6.1 or later
Category	Synchronous API
Header	acexport.h
Static Library	UNIX: libaceclnt.a
Shared Library	Windows: • aceclnt.dll • aceclnt_tcp.dll • sdmsg.dll • xerces-c_3_1_vc80.dll UNIX: • libaceclnt.so • libaceclnt_tcp.so • libxerces-c-3.1.so AIX: • libaceclnt.so **Note:** Rename to libaceclnt.a. For more information, see “Using Dynamic Libraries or Static Libraries in UNIX” on page 38. • libaceclnt_tcp.so • libxerces-c-3.1.a

AceSetPasscode

Description

```
int WINAPI AceSetPasscode(
  SDI_HANDLE hdl,
  char *passcode)
```

The AceSetPasscode function sets the passcode for the specific authentication request associated with the value of the handle passed to it. It is called after a successful return from AceInit and before a call to AceCheck. AceCheck returns with an error if the passcode value has not yet been set.

Architecture

The caller of this synchronous function must supply, as the second argument, a pointer to the character string containing the passcode value to be copied.

Parameters

	Parameter	Description
[in]	hdl	The value of a handle originally assigned by a call to AceInit.
[in]	passcode	A pointer to a character string containing the passcode value.

Return Values

The function returns ACE_SUCCESS if it finds the handle to the authentication data and successfully copies the passcode.

Return value	Description
ACE_ERR_INVALID_HANDLE	Invalid handle, could not locate the data associated with this handle.
ACE_TOO_MANY_CALLERS	An attempt was made to use a handle while a call was already in progress. This happens if a multithreaded application uses the same handle from multiple threads at the same time. This return code is not valid in this version.
ACE_INVALID_ARG	Input parameter Passcode is not valid.

Error Handling

A value of ACE_ERR_INVALID_HANDLE is returned if the function cannot find the handle to the authentication data. A value of ACE_INVALID_ARG is returned if passcode is NULL or its length is too small or too large.

Requirements

Minimum supported Authentication Manager Server Version	6.1 or later
Category	Synchronous API
Header	acexport.h
Static Library	UNIX: libaceclnt.a
Shared Library	Windows: • aceclnt.dll • aceclnt_tcp.dll • sdmsg.dll • xerces-c_3_1_vc80.dll UNIX: • libaceclnt.so • libaceclnt_tcp.so • libxerces-c-3.1.so AIX: • libaceclnt.so **Note:** Rename to libaceclnt.a. For more information, see "Using Dynamic Libraries or Static Libraries in UNIX" on page 38. • libaceclnt_tcp.so • libxerces-c-3.1.a

AceSetPin

Description

```
int WINAPI AceSetPin(
  SDI_HANDLE SdiHandle,
  char *PIN)
```

The AceSetPin function sets the PIN value for the specific authentication request associated with the value of the handle passed to it. It is called after a successful return from AceInit and AceCheck. It is used when the task completed by AceCheck reveals (using a call to AceGetAuthenticationStatus) that New PIN mode is currently enabled for the token. It must be called before calling AceSendPin.

Architecture

This function is synchronous.

Parameters

	Parameter	Description
[in]	hdl	The value of a handle originally assigned by a call to AceInit.
[in]	PIN	A pointer to a character string containing the new PIN value.

Return Values

The function returns ACE_SUCCESS if it finds the handle to the authentication data and successfully copies the new PIN value.

Return value	Description
ACE_ERR_INVALID_HANDLE	Invalid handle, could not locate the data associated with this handle.
ACE_TOO_MANY_CALLERS	An attempt was made to use a handle while a call was already in progress. This happens if a multithreaded application uses the same handle from multiple threads at the same time. This return code is not valid in this version.
ACE_INVALID_ARG	Input parameter PIN is not valid.

Error Handling

A value of ACE_ERR_INVALID_HANDLE is returned if the function cannot find the handle to the authentication data. A value of ACE_INVALID_ARG is returned if the:

- New PIN value is NULL or
- Length is too small or too large or
- Alphanumeric PIN is not supported by Authentication Manager

Requirements

Minimum supported Authentication Manager Server Version	6.1 or later
Category	Synchronous API
Header	acexport.h
Static Library	UNIX: libaceclnt.a
Shared Library	Windows: • aceclnt.dll • aceclnt_tcp.dll • sdmsg.dll • xerces-c_3_1_vc80.dll UNIX: • libaceclnt.so • libaceclnt_tcp.so • libxerces-c-3.1.so AIX: • libaceclnt.so **Note:** Rename to libaceclnt.a. For more information, see "Using Dynamic Libraries or Static Libraries in UNIX" on page 38. • libaceclnt_tcp.so • libxerces-c-3.1.a

AceSetPinCredential

Description

```
int  WINAPI AceSetPinCredential (
  SDI_HANDLE hdl,
  void *pinBuf,
  SD_U32 bufLen )
```

The AceSetPinCredential function sets the PIN value for the specific authentication request associated with the value of the handle passed to it. It is called after a successful return from AceInit and AceCheck. It is used when the task completed by AceCheck reveals (using a call to AceGetAuthenticationStatus) that New PIN mode is currently enabled for the token. It must be called before calling AceSendPin.

AceSetPinCredential is part of the Generic Credentials API. For more information, see "Support for EAP 32 and the Generic Credential API" on page 13.

Note: The credential type must be the same type that was used during the initial authentication request.

Parameters

	Parameter	Description
[in]	hdl	Handle to the current authentication session, which SDInit created.
[in]	pinBuf	The raw new PIN value.
[in]	bufLen	The size of the new PIN data.

Return Values

Return Value	Description
ACE_SUCCESS	The call was successful.
ACE_ERR_INVALID_HANDLE	The handle to the authentication data is incorrect.
ACE_TOO_MANY_CALLERS	An attempt was made to use a handle while a call was already in progress. This happens if a multithreaded application uses the same handle from multiple threads at the same time. This return code is not valid in this version.
ACE_INVALID_ARG	The format of the credential is invalid, or this API is not called during Generic Credential Authentication. Using PIN as NULL also returns this value.

Important: There is a design constraint on the use of SDI_HANDLE. An asynchronous application of the API must avoid concurrent use of the same SDI_HANDLE. Multiple threads using the same SDI_HANDLE return incorrect results.

Error Handling

To handle errors appropriately, use the value returned by this function at a decision point in your code. A successful return allows processing to continue.

Requirements

Minimum supported Authentication Manager Server Version	6.1 or later
Category	Synchronous API
Header	acexport.h
Static Library	UNIX: libaceclnt.a
Shared Library	Windows: • aceclnt.dll • aceclnt_tcp.dll • sdmsg.dll • xerces-c_3_1_vc80.dll UNIX: • libaceclnt.so • libaceclnt_tcp.so • libxerces-c-3.1.so AIX: • libaceclnt.so **Note:** Rename to libaceclnt.a. For more information, see “Using Dynamic Libraries or Static Libraries in UNIX” on page 38. • libaceclnt_tcp.so • libxerces-c-3.1.a

AceSetTimeout

Description

```
int WINAPI AceSetTimeout(
  SDI_HANDLE hdl,
  time_t lifetime,
  void (WINAPI*appCallback)(SDI_HANDLE SdiHandle))
```

The AceSetTimeout function has a dual purpose. When called with a lifetime set to a value other than 0, the function causes the agent to perform AceClose processing automatically. The time is accounted from the time of the last call to any of the API functions. When an appCallBack pointer is passed, the API calls the function after the time-out occurs for performing cleanup operations that must be completed before the handle is discarded.

A secondary purpose of this function is to set up an AceCleanup callback. When the AceCleanup function is called, the callback is used to clean up before the authentication request is discarded. An example of a cleanup callback would be a function that releases any open handles and releases any memory that might have been associated with the authentication as "user data."

The AceSetTimeout function must be called only after a successful return from AceInit, AceStartAuth, or SD_Init.

Note: Do not call AceClose, AceCloseAuth, or SD_Close within the callback function. They are called as part of the time-out handling.

Architecture

This function is synchronous. You can supply a callback, but it is not called unless a time-out occurs or the AceCleanup function is called.

Parameters

	Parameter	Description
[in]	hdl	The value of a handle originally assigned by a call to AceInit.
[in]	lifetime	The time in seconds after which the authentication handle will expire. A value of 0 can be passed if the authentication request is not going to expire.
[in]	CleanupCallback	A pointer to a function written by the developer. This function returns nothing. The handle to the data area specific to the authentication process is passed to this function. A NULL value can be passed instead of a valid function pointer.

Return Values

Return value	Description
ACE_SUCCESS	The handle to the authentication data was found and the lifetime and cleanup callback was successfully set by the function.
ACE_TOO_MANY_CALLERS	An attempt was made to use a handle while a call was already in progress. This happens if a multithreaded application uses the same handle from multiple threads at the same time. This return code is not valid in this version.
ACE_ERR_INVALID_HANDLE	Invalid handle, could not locate the data associated with this handle.

Error Handling

A value of ACE_ERR_INVALID_HANDLE is returned if the handle to the authentication data cannot be found.

Requirements

Minimum supported Authentication Manager Server Version	6.1 or later
Category	Synchronous API
Header	acexport.h
Static Library	UNIX: libaceclnt.a
Shared Library	Windows: • aceclnt.dll • aceclnt_tcp.dll • sdmsg.dll • xerces-c_3_1_vc80.dll UNIX: • libaceclnt.so • libaceclnt_tcp.so • libxerces-c-3.1.so AIX: • libaceclnt.so **Note:** Rename to libaceclnt.a. For more information, see “Using Dynamic Libraries or Static Libraries in UNIX” on page 38. • libaceclnt_tcp.so • libxerces-c-3.1.a

AceSetUserClientAddress

Description

```
int WINAPI AceSetUserClientAddress(
  SDI_HANDLE hdl,
  unsigned char *val)
```

The AceSetUserClientAddress function sets the IP address that is considered the origin of the authentication request processed by a subsequent call to AceClientCheck. A call to this function must be made before calling AceClientCheck. AceClientCheck performs an authentication check just as AceCheck does. However, this request is done on behalf of the client machine whose IP address is set by a call to the AceSetUserClientAddress function.

The AceSetUserClientAddress function is called only after a successful return from AceInit.

This function is useful in situations involving proxies and is part of the Generic Credentials API. For more information, see "Support for EAP 32 and the Generic Credential API" on page 13.

Note: AceSetClientAddr is not explained separately in this guide. It also performs the same functionality as AceSetUserClientAddress. If your agent is already using AceSetClientAddr, you do not need to modify your code.

Architecture

This function is synchronous.

Parameters

	Parameter	Description
[in]	hdl	The value of a handle originally assigned by a call to AceInit.
[in]	val	A pointer to a character array containing the IP address value. The IP address must be the full address arranged in network byte order.

Return Values

Return value	Description
ACE_SUCCESS	The handle to the authentication data was found and the IP address value was successfully copied by the function.

ACE_ERR_INVALID_HANDLE	Invalid handle, could not locate the data associated with this handle.
ACE_TOO_MANY_CALLERS	An attempt was made to use a handle while a call was already in progress. This happens if a multithreaded application uses the same handle from multiple threads at the same time. This return code is not valid in this version.
ACE_INVALID_ARG	Input parameter val is not valid.

Error Handling

A value of ACE_ERR_INVALID_HANDLE is returned if the handle to the authentication data cannot be found.

Requirements

Minimum supported Authentication Manager Server Version	6.1 or later
Category	Synchronous API
Header	acexport.h
Static Library	UNIX: libaceclnt.a
Shared Library	Windows: • aceclnt.dll • aceclnt_tcp.dll • sdmsg.dll • xerces-c_3_1_vc80.dll UNIX: • libaceclnt.so • libaceclnt_tcp.so • libxerces-c-3.1.so AIX: • libaceclnt.so **Note:** Rename to libaceclnt.a. For more information, see “Using Dynamic Libraries or Static Libraries in UNIX” on page 38. • libaceclnt_tcp.so • libxerces-c-3.1.a

AceSetUserData

Description

```
int WINAPI AceSetUserData(
  SDI_HANDLE SdiHandle,
  unsigned int userData)
```

The AceSetUserData function sets the userData argument for the specific authentication request associated with the value of the handle passed to it. It is called after a successful return from AceInit, AceStartAuth, or SD_Init. The data set by this function can be any 32-bit quantity, such as a pointer. It can be used to override the user data set by the AceInit call, or to supply similar data for the AceStartAuth or SD_Init calls. The data set by this call can be retrieved later by a call to AceGetUserData.

The purpose of the calls AceSetUserData and AceGetUserData is to supply agent developers with a convenient location and associated routines for storing and retrieving data related to the authentication in progress. Although this is mostly useful in cases of true asynchronous operation, it is available to both the synchronous calls and the asynchronous calls.

An example of a use for these calls is in an application that operates within a server. The application might have to maintain information about the user authentication. In that case, the code would allocate some memory and pass a pointer to the memory as the 32-bit data reference to AceSetUserData. When this data is needed at a later point in the authentication process, the code only maintains the SDI_HANDLE value.

To access the data, the code would then call AceGetUserData to retrieve the pointer to the original data and process it. The caller of these functions is responsible for the disposition of any resources that are associated with the data referenced by the 32-bit value. The Authentication API furnishes the value whenever it is requested.

Architecture

The caller of this synchronous function must supply, as the second argument, any 32-bit value to be copied.

Parameters

	Parameter	Description
[in]	hdl	The value of a handle originally assigned by a call to AceInit.
[in]	val	A 32-bit value that holds data to be associated with the authentication request. It can be passed a handle or any other pointer type.

Return Values

Return value	Description
ACE_SUCCESS	The handle to the authentication data was found and the value of userData was successfully copied by the function.
ACE_ERR_INVALID_HANDLE	Invalid handle, could not locate the data associated with this handle.
ACE_TOO_MANY_CALLERS	An attempt was made to use a handle while a call was already in progress. This happens if a multithreaded application uses the same handle from multiple threads at the same time. This return code is not valid in this version.

Error Handling

A value of ACE_ERR_INVALID_HANDLE is returned if the handle to the authentication data cannot be found.

Requirements

Minimum supported Authentication Manager Server Version	6.1 or later
Category	Synchronous API
Header	acexport.h
Static Library	UNIX: libaceclnt.a

Minimum supported Authentication Manager Server Version	6.1 or later
Shared Library	Windows: • aceclnt.dll • aceclnt_tcp.dll • sdmsg.dll • xerces-c_3_1_vc80.dll UNIX: • libaceclnt.so • libaceclnt_tcp.so • libxerces-c-3.1.so AIX: • libaceclnt.so **Note:** Rename to libaceclnt.a. For more information, see "Using Dynamic Libraries or Static Libraries in UNIX" on page 38. • libaceclnt_tcp.so • libxerces-c-3.1.a

AceSetUsername

Description

```
int WINAPI AceSetUsername(
  SDI_HANDLE hdl,
  char *username)
```

The AceSetUsername function sets the user name argument for the specific authentication request associated with the value of the handle passed to it. It is called after a successful return from AceInit, but before a call to AceCheck. AceCheck returns with an error if this value has not yet been set.

Architecture

The caller of this synchronous function must supply as the second argument a pointer to the character string containing the user name value to be copied.

Parameters

	Parameter	Description
[in]	hdl	The value of a handle originally assigned by a call to AceInit.
[in]	username	A pointer to the NULL terminated user name string. The user name can be a maximum of 48 characters if using the Authentication Manager 6.1 series, and a maximum of 64 characters if using the Authentication Manager 7.1 series and maximum of 255 characters if using Authentication Manager 8.

Return Values

Return value	Description
ACE_SUCCESS	The handle to the authentication data was found and the user name was successfully copied by the function.
ACE_ERR_INVALID_HANDLE	Invalid handle, could not locate the data associated with this handle.
ACE_TOO_MANY_CALLERS	An attempt was made to use a handle while a call was already in progress. This happens if a multithreaded application uses the same handle from multiple threads at the same time. This return code is not valid in this version.
ACE_INVALID_ARG	Input parameter username is not valid.

Error Handling

A value of ACE_ERR_INVALID_HANDLE is returned if the handle to the authentication data cannot be found. A value of ACE_INVALID_ARG is returned if username is NULL or its length is too large.

Requirements

Minimum supported Authentication Manager Server Version	6.1 or later
Category	Synchronous API
Header	acexport.h
Static Library	UNIX: libaceclnt.a
Shared Library	Windows: • aceclnt.dll • aceclnt_tcp.dll • sdmsg.dll • xerces-c_3_1_vc80.dll UNIX: • libaceclnt.so • libaceclnt_tcp.so • libxerces-c-3.1.so AIX: • libaceclnt.so **Note:** Rename to libaceclnt.a. For more information, see "Using Dynamic Libraries or Static Libraries in UNIX" on page 38. • libaceclnt_tcp.so • libxerces-c-3.1.a

AceShutdown

Description

```
SD_BOOL WINAPI AceShutdown(
  void (WINAPI*appCallback)(SDI_HANDLE));
```

The AceShutdown function can be called when the application has completed its function and is about to terminate. After AceShutdown has been called, your code must call AceInitialize again before making any calls to the library. All calls return ACE_ERR_NOT_INITIALIZED after AceShutdown completes. After this call has returned, every handle that has been returned from a call to AceInit, AceStartAuth, or SD_Init becomes invalid.

The callback argument is used to clean up before the authentication handle is discarded. If a callback has been defined by a call to AceSetTimeout, it is not called unless the callback for AceShutdown has a value other than NULL. In this case, the callback passed as an argument to AceShutdown is used instead.

Architecture

This synchronous function views the internal list of outstanding authentication handles and performs the cleanup operation on them. This activity behaves as if the user's actions resulted in the appropriate final call (AceClose, AceCloseAuth, or SD_Close) being made. In addition, both the internal worker threads and any memory or other resources are released.

Parameters

	Parameter	Description
[in]	CleanupCallback	A pointer to a function written by the developer. This function returns nothing. The handle to the data area specific to the authentication process is passed to this function. A NULL value can be passed instead of a valid function pointer.

Return Values

The function returns SD_TRUE if it is successful and SD_FALSE if it fails. Upon return, every outstanding authentication has been discarded. All handles that have been returned by AceInit, AceStartAuth, or SD_Init become invalid.

Error Handling

If the function is called more than once, or before any calls to AceInitialize, it returns SD_FALSE.

Requirements

Minimum supported Authentication Manager Server Version	6.1 or later
Category	Synchronous API
Header	acexport.h
Static Library	UNIX: libaceclnt.a
Shared Library	Windows: • aceclnt.dll • aceclnt_tcp.dll • sdmsg.dll • xerces-c_3_1_vc80.dll UNIX: • libaceclnt.so • libaceclnt_tcp.so • libxerces-c-3.1.so AIX: • libaceclnt.so **Note:** Rename to libaceclnt.a. For more information, see “Using Dynamic Libraries or Static Libraries in UNIX” on page 38. • libaceclnt_tcp.so • libxerces-c-3.1.a

AceStartAuth

Description

```
SD_ERROR WINAPI AceStartAuth(
  LP_SDI_HANDLE SdiHandle,
  char *userID,
  SD_I32 userIDLen,
  SD_BOOL *moreData,
  SD_BOOL *echoFlag,
  SD_I32 *respTimeout,
  SD_I32 *nextRespLen,
  char *promptStr,
  SD_I32 *promptStrLen)
```

The AceStartAuth function is designed to be used with AceContinueAuth and AceCloseAuth. These three functions provide a high-level model of the protocol and process of authenticating users. The functions can be used with any number of challenge or response protocols.

The values of the prompts returned by this function are stored in a message catalog that can be modified by an agent developer. For more information, see Appendix C, "Modifying the Message Catalog."

Note: If you use this version of the RSA Authentication Agent API with pre-5.0 agents, you do not have to change your use of the AceStartAuth, AceContinueAuth, and AceCloseAuth functions to support two-step authentication. Two-step authentication is automatically performed within these three functions.

Architecture

The AceStartAuth function provides the first step in authenticating with the RSA SecurID protocol using the synchronous API.

If the function returns successfully, then your code calls AceContinueAuth with the required passcode. Your code continues to call AceContinueAuth until no more data is required by the authentication context, or an error occurs.

Whenever AceStartAuth returns successfully, a call to AceCloseAuth must eventually be made for the specific authentication context.

Parameters

	Parameter	Description
[out]	hdl	A pointer to an SDI_HANDLE whose value is specified by the function.
[in]	userID	A pointer to the string containing the user name value.
[in]	userIDLen	An integer containing the length of the userID string.
[out]	moreData	A flag that is set by the API to indicate whether more data is needed by the authentication context.
[out]	noEcho	A flag that guides the developer as to whether the next expected response is echoed to the screen.
[out]	respTimeout	A hint to the developer about how long to display the next prompt string for the user.
[out]	nextRespLen	An indicator of the maximum number of bytes of data expected in the next developer-supplied response.
[out]	promptStr	A developer-supplied character array to be filled in by the API with the string that the caller uses as the next message displayed to the user.
[in, out]	promptStrLen	The size of the developer-supplied storage for the promptStr; its value becomes the length of the filled-in promptStr string.

Return Values

The AceStartAuth function, if successful, returns ACM_OK. Otherwise, it returns with an error. Your code must call AceCloseAuth only if the AceStartAuth function is successful. Common error values are:

Return Value	Description
ACE_INVALID_ARG	Could not set the userID value.
ACE_NOT_ENOUGH_STORAGE	The size of the developer-supplied promptStr is too small. PromptStr reads "Program error: buffer too small." If the developer-supplied array is too small even for this error message, the error message is truncated to match the promptLen supplied.
ACE_CFGFILE_NOT_FOUND	The **sdconf.rec** file could not be opened.
ACE_CFGFILE_READ_FAIL	Unable to read **sdconf.rec** file.

ACE_TOO_MANY_CALLERS	An attempt was made to use a handle while a call was already in progress. This happens if a multithreaded application uses the same handle from multiple threads at the same time. This return code is not valid in this version.

Error Handling

Your code does not have to call AceCloseAuth if the AceStartAuth function is unsuccessful.

Requirements

Minimum supported Authentication Manager Server Version	6.1 or later
Category	Synchronous API
Header	acexport.h
Static Library	UNIX: libaceclnt.a
Shared Library	Windows: • aceclnt.dll • aceclnt_tcp.dll • sdmsg.dll • xerces-c_3_1_vc80.dll UNIX: • libaceclnt.so • libaceclnt_tcp.so • libxerces-c-3.1.so AIX: • libaceclnt.so **Note:** Rename to libaceclnt.a. For more information, see "Using Dynamic Libraries or Static Libraries in UNIX" on page 38. • libaceclnt_tcp.so • libxerces-c-3.1.a

SD_Check

Description

```
int SD_Check(
  SDI_HANDLE SdiHandle,
  char *passcode,
  char *username)
```

SD_Check performs authentication by checking the validity of the passcode entered by a user. SD_Check must be called only after successfully calling SD_Init. The integrating application must perform all I/O because SD_Check does not display the authentication prompts and messages.

This function is a synchronous wrapper for AceCheck, AceSetUsername, and AceSetPasscode.

Parameters

	Parameter	Description
[in]	hdl	The value of a handle originally assigned by a call to SD_Init.
[in]	passcode	A pointer to the NULL terminated passcode string.
[in]	username	A pointer to the NULL terminated user name string. The user name can be a maximum of 48 characters if using the Authentication Manager 6.1 series, and a maximum of 64 characters if using the Authentication Manager 7.1 series and maximum of 255 characters if using Authentication Manager 8.

Return Values

Note: This function returns authentication result codes as return values.

Return Value	Description
ACM_OK	The user successfully authenticated. You can use the AceGetShell function to access the shell field.
ACE_ERR_INVALID_HANDLE	The handle value is invalid.
ACM_ACCESS_DENIED	The user failed authentication.

ACM_NEXT_CODE_REQUIRED	Next tokencode required. Use SD_Next to complete the transaction. This API must be called within 140 seconds if using Authentication Manager 6.1.2, or 120 seconds if using Authentication Manager 7.1. If using Authentication Manager 8.0, the timeout is 180 seconds.
ACM_NEW_PIN_REQUIRED	New PIN required. You can use the AceGetxxx functions to access the PIN limits. Use SD_Pin to complete the transaction. These APIs must be called within 140 seconds if using Authentication Manager 6.1.2, or 120 seconds if using Authentication Manager 7.1. If using Authentication Manager 8.0, the timeout is 180 seconds.
ACE_TOO_MANY_CALLERS	An attempt was made to use a handle while a call was already in progress. This happens if a multithreaded application uses the same handle from multiple threads at the same time. This return code is not valid in this version.
ACE_INVALID_ARG	An invalid argument was supplied to the API function.

Requirements

Minimum supported Authentication Manager Server Version	6.1 or later
Category	Synchronous API
Header	acexport.h
Static Library	UNIX: libaceclnt.a
Shared Library	Windows: • aceclnt.dll • aceclnt_tcp.dll • sdmsg.dll • xerces-c_3_1_vc80.dll UNIX: • libaceclnt.so • libaceclnt_tcp.so • libxerces-c-3.1.so AIX: • libaceclnt.so **Note:** Rename to libaceclnt.a. For more information, see "Using Dynamic Libraries or Static Libraries in UNIX" on page 38. • libaceclnt_tcp.so • libxerces-c-3.1.a

SD_CheckCredential

Description

```
int WINAPI SD_CheckCredential (
  SDI_HANDLE hdl,
  SD_CHAR *username,
  RSA_AUTH_CRED_TYPE credType,
  void *credBuf,
  SD_U32 credLen )
```

The SD_CheckCredential function performs an authentication with the given credential. SD_CheckCredential is part of the Generic Credentials API. For more information, see "Support for EAP 32 and the Generic Credential API" on page 13.

Parameters

	Parameter	Description
[in]	hdl	Handle to the current authentication session, which SDInit created.
[in]	username	Null-terminated string username. The user name can be a maximum of 48 characters if using the Authentication Manager 6.1 series, and a maximum of 64 characters if using the Authentication Manager 7.1 series and maximum of 255 characters if using the Authentication Manager 8.
[in]	credType	Specifies the generic credential type.
[in]	credBuf	Raw credential data.
[in]	credLen	Size of credential data.

Return Values

Return Value	Description
ACM_OK	User successfully authenticated.
ACM_ACCESS_DENIED	User failed authentication, or node secret file is missing.
ACM_NEXT_CODE_REQUIRED	Next tokencode required, then issue SD_NextCredential() to complete the transaction. This API must be called within 140 seconds if using Authentication Manager 6.1.2, or 120 seconds if using Authentication Manager 7.1. If using Authentication Manager 8.0, the timeout is 180 seconds.

ACM_NEW_PIN_REQUIRED	New PIN required, then issue SD_PinCredential() to complete the transaction. This API must be called within 140 seconds if using Authentication Manager 6.1.2, or 120 seconds if using Authentication Manager 7.1. If using Authentication Manager 8.0, the timeout is 180 seconds.
ACE_ERR_INVALID_HANDLE	Invalid handle, could not locate the data associated with this handle.
ACE_TOO_MANY_CALLERS	An attempt was made to use a handle while a call was already in progress. This happens if a multithreaded application uses the same handle from multiple threads at the same time. This return code is not valid in this version.
ACE_INVALID_ARG	An invalid argument was supplied to the API function.

Important: There is a design constraint on the use of SDI_HANDLE. An asynchronous application of the API must avoid concurrent use of the same SDI_HANDLE. Multiple threads using the same SDI_HANDLE return incorrect results.

Error Handling

To handle errors appropriately, use the value returned by this function at a decision point in your code. A successful return allows processing to continue.

Requirements

Minimum supported Authentication Manager Server Version	6.1 or later
Category	Synchronous API
Header	acexport.h
Static Library	UNIX: libaceclnt.a
Shared Library	Windows: • aceclnt.dll • aceclnt_tcp.dll • sdmsg.dll • xerces-c_3_1_vc80.dll UNIX: • libaceclnt.so • libaceclnt_tcp.so • libxerces-c-3.1.so AIX: • libaceclnt.so **Note:** Rename to libaceclnt.a. For more information, see "Using Dynamic Libraries or Static Libraries in UNIX" on page 38. • libaceclnt_tcp.so • libxerces-c-3.1.a

SD_ClientCheck

Description

```
int SD_ClientCheck(
  SDI_HANDLE SdiHandle,
  char *passcode,
  char *username,
  unsigned long client_addr)
```

SD_ClientCheck performs authentication by checking the validity of the passcode entered by a user. SD_ClientCheck must be called only after successfully calling SD_Init. The integrating application must perform all I/O because SD_ClientCheck does not display the authentication prompts and messages.

This synchronous function wraps the functionality provided by AceSetUserName, AceSetPasscode, AceSetUserClientAddress, and AceClientCheck.

Note: This API should not be used for IPv6 in the current SDK.

Parameters

	Parameter	Description
[in]	hdl	The value of a handle originally assigned by a call to SD_Init.
[in]	passcode	A pointer to the NULL-terminated passcode string. The passcode must contain 4 to 16 characters.
[in]	username	Null-terminated string user name. The user name can be a maximum of 48 characters if using the Authentication Manager 6.1 series, a maximum of 64 characters if using the Authentication Manager 7.1 series, and a maximum of 255 characters if using Authentication Manager 8.0.
[in]	cli_addr	A pointer to an IP address value.

Return Values

Note: This function returns authentication result codes as return values.

Return Value	Description
ACM_OK	The user successfully authenticated. You can use the AceGetShell function to determine the user's shell.

ACE_ERR_INVALID_HANDLE	The handle value is invalid.
ACM_ACCESS_DENIED	The user failed to authenticate successfully.
ACM_NEXT_CODE_REQUIRED	Next tokencode required. Use SD_NextCredential to complete the transaction. This API must be called within 140 seconds if using Authentication Manager 6.1.2, or 120 seconds if using Authentication Manager 7.1. If using Authentication Manager 8.0, the timeout is 180 seconds.
ACM_NEW_PIN_REQUIRED	New PIN required. You can use the AceGetxxx functions to get access to the PIN limits. Use SD_PinCredential to complete the transaction. These APIs must be called within 140 seconds if using Authentication Manager 6.1.2, or 120 seconds if using Authentication Manager 7.1. If using Authentication Manager 8.0, the timeout is 180 seconds.
ACE_TOO_MANY_CALLERS	An attempt was made to use a handle while a call was already in progress. This happens if a multithreaded application uses the same handle from multiple threads at the same time. This return code is not valid in this version.
ACE_INVALID_ARG	An invalid argument was supplied to the API function.

Requirements

Minimum supported Authentication Manager Server Version	6.1 or later
Category	Synchronous API
Header	acexport.h
Static Library	UNIX: libaceclnt.a
Shared Library	Windows: • aceclnt.dll • aceclnt_tcp.dll • sdmsg.dll • xerces-c_3_1_vc80.dll UNIX: • libaceclnt.so • libaceclnt_tcp.so • libxerces-c-3.1.so AIX: • libaceclnt.so **Note:** Rename to libaceclnt.a. For more information, see “Using Dynamic Libraries or Static Libraries in UNIX” on page 38. • libaceclnt_tcp.so • libxerces-c-3.1.a

SD_ClientCheckCredential

Description

```
int WINAPI SD_ClientCheckCredential (
  SDI_HANDLE hdl,
  SD_CHAR *username,
  RSA_AUTH_CRED_TYPE credType,
  void *credBuf,
  SD_U32 credLen,
  SD_U32 cli_addr )
```

The SD_ClientCheckCredential performs an authentication with the given credential on behalf of a particular host. SD_ClientCheckCredential is part of the Generic Credentials API. For more information, see "Support for EAP 32 and the Generic Credential API" on page 13.

Note: This API should not be used for IPv6 in the current SDK.

Parameters

	Parameter	Description
[in]	hdl	Handle to the current authentication session, which SDInit created.
[in]	username	Null-terminated string user name. The user name can be a maximum of 48 characters if using the Authentication Manager 6.1 series, a maximum of 64 characters if using the Authentication Manager 7.1 series, and a maximum of 255 characters if using Authentication Manager 8.0.
[in]	credType	Specifies the generic credential type.
[in]	credBuf	Raw credential data.
[in]	credLen	Size of credential data.
[in]	cli_addr	Client IP address.

Return Values

The value returned by this function is one of the following.

Return Value	Description
ACM_OK	User successfully authenticated.

ACM_ACCESS_DENIED	User failed authentication or node secret file is missing.
ACM_NEXT_CODE_REQUIRED	Next tokencode required, then issue SD_NextCredential() to complete the transaction. This API must be called within 140 seconds if using Authentication Manager 6.1.2, or 120 seconds if using Authentication Manager 7.1. If using Authentication Manager 8.0, the timeout is 180 seconds.
ACM_NEW_PIN_REQUIRED	New PIN required, then issue SD_PinCredential() to complete the transaction. This API must be called within 140 seconds if using Authentication Manager 6.1.2, or 120 seconds if using Authentication Manager 7.1. If using Authentication Manager 8.0, the timeout is 180 seconds.
ACE_ERR_INVALID_HANDLE	Invalid handle, could not locate the data associated with this handle.
ACE_TOO_MANY_CALLERS	An attempt was made to use a handle while a call was already in progress. This happens if a multithreaded application uses the same handle from multiple threads at the same time. This return code is not valid in this version.
ACE_INVALID_ARG	An invalid argument was supplied to the API function.
ACE_UNDEFINED_CLIENTADDR	The client IP address was not set.

Important: There is a design constraint on the use of SDI_HANDLE. An asynchronous application of the API must avoid concurrent use of the same SDI_HANDLE. Multiple threads using the same SDI_HANDLE return incorrect results.

Error Handling

To handle errors appropriately, use the value returned by this function at a decision point in your code. A successful return allows processing to continue.

Requirements

Minimum supported Authentication Manager Server Version	6.1 or later
Category	Synchronous API
Header	acexport.h
Static Library	UNIX: libaceclnt.a
Shared Library	Windows: • aceclnt.dll • aceclnt_tcp.dll • sdmsg.dll • xerces-c_3_1_vc80.dll UNIX: • libaceclnt.so • libaceclnt_tcp.so • libxerces-c-3.1.so AIX: • libaceclnt.so **Note:** Rename to libaceclnt.a. For more information, see "Using Dynamic Libraries or Static Libraries in UNIX" on page 38. • libaceclnt_tcp.so • libxerces-c-3.1.a

SD_Close

Description

```
int SD_Close(
  SDI_HANDLE SdiHandle)
```

SD_Close releases any resources allocated by SD_Init. This function is called after a successful call to SD_Init.

This function is a synchronous wrapper for AceClose.

Parameters

	Parameter	Description
[in]	hdl	The value of a handle originally assigned by a call to SD_Init.

Return Values

Note: This function returns authentication result codes as return values.

Return Value	Description
ACM_OK	The authentication session has closed successfully.
ACE_ERR_INVALID_HANDLE	Handle value is invalid.

Requirements

Minimum supported Authentication Manager Server Version	6.1 or later
Category	Synchronous API
Header	acexport.h
Static Library	UNIX: libaceclnt.a
Shared Library	Windows: • aceclnt.dll • aceclnt_tcp.dll • sdmsg.dll • xerces-c_3_1_vc80.dll UNIX: • libaceclnt.so • libaceclnt_tcp.so • libxerces-c-3.1.so AIX: • libaceclnt.so **Note:** Rename to libaceclnt.a. For more information, see "Using Dynamic Libraries or Static Libraries in UNIX" on page 38. • libaceclnt_tcp.so • libxerces-c-3.1.a

SD_Init

Description

```
int SD_Init(
  LP_SDI_HANDLE pSdiHandle);
```

SD_Init initializes communication between the agent and the RSA Authentication Manager. It initializes the socket and makes a call to the RSA Authentication Manager to verify communication. SD_Init must be called after a successful call to AceInitialize, but before calling any other API function.

This function is a synchronous wrapper for AceInit.

Parameters

	Parameter	Description
[out]	pSdiHandle	A pointer to a handle that is assigned by the SD_Init function.

Return Values

Return Value	Description
ACM_OK	No error.
ACE_CFGFILE_NOT_FOUND	The **sdconf.rec** file could not be opened.
ACE_CFGFILE_READ_FAIL	Unable to read **sdconf.rec** file.
ACM_NO_SERVER	Failed to initialize, Authentication Manager is not responding.

Error Handling

If SD_Init returns ACM_OK, you must call SD_Close to release the resources associated with the authentication handle. If any error occurs, the handle is not set, and you do not need to call SD_Close.

Requirements

Minimum supported Authentication Manager Server Version	6.1 or later
Category	Synchronous API
Header	acexport.h
Static Library	UNIX: libaceclnt.a
Shared Library	Windows: • aceclnt.dll • aceclnt_tcp.dll • sdmsg.dll • xerces-c_3_1_vc80.dll UNIX: • libaceclnt.so • libaceclnt_tcp.so • libxerces-c-3.1.so AIX: • libaceclnt.so **Note:** Rename to libaceclnt.a. For more information, see "Using Dynamic Libraries or Static Libraries in UNIX" on page 38. • libaceclnt_tcp.so • libxerces-c-3.1.a

SD_InitEx

Description

```
int WINAPI SD_InitEx(
    LP_SDI_HANDLE          SdiHandle,
    SD_BOOL                bSupportOA,
    RSA_DA_AGENT_TYPEagentType)
```

SD_InitEx performs the same initialization tasks as SD_Init and, additionally, initializes communication with the specified offline service.

Parameters

	Parameter	Description
[out]	hdl	A pointer to a handle that is assigned upon return from the function.
[in]	bSupportOA	A flag to indicate whether to contact the offline service. This flag is no longer supported in this version and will be ignored.
[in]	agentType	Indicates which offline service to communicate with. This flag is no longer supported in this version and will be ignored.

Return Values

Return Value	Description
ACM_OK	Indicates that RSA Authentication Manager is available or, if not, the offline service is available.
ACE_INIT_NO_RESOURCE	A low memory condition caused SD_InitEx to fail.
ACE_CFGFILE_NOT_FOUND	The **sdconf.rec** file could not be opened.
ACE_CFGFILE_READ_FAIL	Unable to read **sdconf.rec** file.
ACM_NO_SERVER	Failed to initialize, Authentication Manager is not responding.

Requirements

Minimum supported Authentication Manager Server Version	6.1 or later
Category	Synchronous API
Header	acexport.h
Static Library	UNIX: libaceclnt.a
Shared Library	Windows: • aceclnt.dll • aceclnt_tcp.dll • sdmsg.dll • xerces-c_3_1_vc80.dll UNIX: • libaceclnt.so • libaceclnt_tcp.so • libxerces-c-3.1.so AIX: • libaceclnt.so **Note:** Rename to libaceclnt.a. For more information, see “Using Dynamic Libraries or Static Libraries in UNIX” on page 38. • libaceclnt_tcp.so • libxerces-c-3.1.a

SD_Lock

Note: This function is not supported in this release of the API

Description

```
int SD_Lock(
  SDI_HANDLE SdiHandle,
  char *username)
```

SD_Lock performs the operations of the AceSetUserName and AceLock functions. SD_Lock is called only after successfully calling SD_Init.

For information about using this function to help prevent unauthorized authentications, see "Load Balancing" on page 14.

Parameters

	Parameter	Description
[in]	hdl	The value of a handle originally assigned by a call to SD_Init.
[in]	username	A pointer to the NULL-terminated username string. The user name can be a maximum of 48 characters if using the Authentication Manager 6.1 series, and a maximum of 64 characters if using the Authentication Manager 7.1 series.

Return Values

Note: This function returns authentication result codes as return values.

Return Value	Description
ACM_OK	The user name lock request has been sent to the RSA Authentication Manager. Use the SD_Check function to process the passcode.
ACE_ERR_INVALID_HANDLE	The handle value is invalid.
ACM_ACCESS_DENIED	Communication with the RSA Authentication Manager has failed. The failure might result due to a communication time-out or a mismatch in the node secret between the agent and the server. Work with the RSA Authentication Manager administrator to troubleshoot the problem.

Return Value	Description
ACE_TOO_MANY_CALLERS	An attempt was made to use a handle while a call was already in progress. This happens if a multithreaded application uses the same handle from multiple threads at the same time. This return code is not valid in this version.
ACE_INVALID_ARG	An invalid argument was supplied to the API function.

Requirements

Minimum supported Authentication Manager Server Version	6.1
Category	Synchronous API
Header	acexport.h
Static Library	UNIX: libaceclnt.a
Shared Library	Windows: • aceclnt.dll • aceclnt_tcp.dll • sdmsg.dll • xerces-c_3_1_vc80.dll UNIX: • libaceclnt.so • libaceclnt_tcp.so • libxerces-c-3.1.so AIX: • libaceclnt.so **Note:** Rename to libaceclnt.a. For more information, see "Using Dynamic Libraries or Static Libraries in UNIX" on page 38. • libaceclnt_tcp.so • libxerces-c-3.1.a

SD_Next

Description

```
int SD_Next(
  SDI_HANDLE SdiHandle,
  char *nextcode)
```

SD_Next performs the Next Code operation, which takes a second successive tokencode from a user and checks its validity. SD_Next is called only in response to an ACM_NEXT_CODE_ REQUIRED return from SD_Check. The integrating application must perform all I/O because SD_Next does not display the Next Code prompt.

This function is a synchronous wrapper for AceSetNextPasscode, and AceSendNextPasscode.

Parameters

	Parameter	Description
[in]	hdl	The value of a handle originally assigned by a call to SD_Init.
[in]	nextcode	A pointer to the NULL terminated passcode string.

Return Values

Note: This function returns authentication result codes as return values.

Return Value	Description
ACM_OK	The user successfully authenticated. You can use the AceGetShell function to access the shell field.
ACE_ERR_INVALID_HANDLE	The handle value is invalid.
ACM_ACCESS_DENIED	The user failed authentication.
ACE_TOO_MANY_CALLERS	An attempt was made to use a handle while a call was already in progress. This happens if a multithreaded application uses the same handle from multiple threads at the same time. This return code is not valid in this version.
ACE_INVALID_ARG	An invalid argument was supplied to the API function.

Requirements

Minimum supported Authentication Manager Server Version	6.1 or later
Category	Synchronous API
Header	acexport.h
Static Library	UNIX: libaceclnt.a
Shared Library	Windows: • aceclnt.dll • aceclnt_tcp.dll • sdmsg.dll • xerces-c_3_1_vc80.dll UNIX: • libaceclnt.so • libaceclnt_tcp.so • libxerces-c-3.1.so AIX: • libaceclnt.so **Note:** Rename to libaceclnt.a. For more information, see “Using Dynamic Libraries or Static Libraries in UNIX” on page 38. • libaceclnt_tcp.so • libxerces-c-3.1.a

SD_NextCredential

Description

```
int WINAPI SD_NextCredential (
  SDI_HANDLE hdl,
  void *nextBuf,
  SD_U32 bufLen )
```

The SD_NextCredential sends the next generic credential, which must be the same credential type used for the initial authentication attempt. SD_NextCredential is part of the Generic Credentials API. For more information, see "Support for EAP 32 and the Generic Credential API" on page 13.

Parameters

	Parameter	Description
[in]	hdl	Handle to the current authentication session, which SDInit created.
[in]	nextBuf	Raw next credential data.
[in]	bufLen	Size of next credential data.

Return Values

Return Value	Description
ACM_OK	User successfully authenticated.
ACM_ACCESS_DENIED	User failed authentication, or node secret file is missing.
ACE_ERR_INVALID_HANDLE	Invalid handle; could not locate the data associated with this handle.
ACE_TOO_MANY_CALLERS	An attempt was made to use a handle while a call was already in progress. This happens if a multithreaded application uses the same handle from multiple threads at the same time. This return code is not valid in this version.
ACE_INVALID_ARG	Input parameter not valid.

Important: There is a design constraint on the use of SDI_HANDLE. An asynchronous application of the API must avoid concurrent use of the same SDI_HANDLE. Multiple threads using the same SDI_HANDLE return incorrect results.

Error Handling

To handle errors appropriately, use the value returned by this function at a decision point in your code. A successful return allows processing to continue.

Requirements

Minimum supported Authentication Manager Server Version	6.1 or later
Category	Synchronous API
Header	acexport.h
Static Library	UNIX: libaceclnt.a
Shared Library	Windows: • aceclnt.dll • aceclnt_tcp.dll • sdmsg.dll • xerces-c_3_1_vc80.dll UNIX: • libaceclnt.so • libaceclnt_tcp.so • libxerces-c-3.1.so AIX: • libaceclnt.so **Note:** Rename to libaceclnt.a. For more information, see “Using Dynamic Libraries or Static Libraries in UNIX” on page 38. • libaceclnt_tcp.so • libxerces-c-3.1.a

SD_Pin

Description

```
int SD_Pin(
  SDI_HANDLE SdiHandle,
  char *pin)
```

SD_Pin performs the New PIN operation in which a new PIN is transmitted to the RSA Authentication Manager for storage in a token record. SD_Pin is called only in response to an ACM_NEW_PIN_REQUIRED return from SD_Check. The integrating application must perform all I/O, because SD_Pin does not display the New PIN prompts and messages.

Note: Do not consider users to be authenticated as soon as they have completed the New PIN operation. Users must use the new PIN to be authenticated.

Parameters

	Parameter	Description
[in]	hdl	The value of a handle originally assigned by a call to SD_Init.
[in]	pin	A pointer to the NULL terminated PIN string. The PIN must contain 4 to 8 characters. If you pass a NULL pointer or an empty string, it results in an aborted new PIN process, which is equivalent to the AceCancelPin function.

Return Values

Note: This function returns authentication result codes as return values.

Return Value	Description
ACE_INVALID_ARG	The pin argument is an invalid length, or it does not meet the PIN policy set by Authentication Manager. Use AceGetPinParams to get the PIN policy set by Authentication Manager.
ACE_ERR_INVALID_HANDLE	The handle value is invalid.
ACE_TOO_MANY_CALLERS	An attempt was made to use a handle while a call was already in progress. This happens if a multithreaded application uses the same handle from multiple threads at the same time. This return code is not valid in this version.

Return Value	Description
ACM_NEW_PIN_ACCEPTED	The RSA Authentication Manager has accepted the new PIN. The user is now required to authenticate with the new PIN.
ACM_NEW_PIN_REJECTED	The RSA Authentication Manager rejected the new PIN.

Requirements

Minimum supported Authentication Manager Server Version	6.1 or later
Category	Synchronous API
Header	acexport.h
Static Library	UNIX: libaceclnt.a
Shared Library	Windows: • aceclnt.dll • aceclnt_tcp.dll • sdmsg.dll • xerces-c_3_1_vc80.dll UNIX: • libaceclnt.so • libaceclnt_tcp.so • libxerces-c-3.1.so AIX: • libaceclnt.so **Note:** Rename to libaceclnt.a. For more information, see "Using Dynamic Libraries or Static Libraries in UNIX" on page 38. • libaceclnt_tcp.so • libxerces-c-3.1.a

SD_PinCredential

Description

```
int WINAPI SD_PinCredential (
  SDI_HANDLE hdl,
  void *pinBuf,
  SD_U32 bufLen )
```

SD_PinCredential sends the new PIN. The credential type must be the same credential type used for the initial authentication attempt. SD_PinCredential is part of the Generic Credentials API. For more information, see "Support for EAP 32 and the Generic Credential API" on page 13.

Parameters

[in]	hdl	Handle to the current authentication session, which SDInit created.
[in]	pinBuf	Raw new pin data.
[in]	bufLen	Size of new pin data.

Return Values

Return Value	Description
ACM_NEW_PIN_ACCEPTED	The RSA Authentication Manager accepted the new PIN.
ACM_NEW_PIN_REJECTED	The RSA Authentication Manager rejected the new PIN.
ACE_ERR_INVALID_HANDLE	Invalid handle, could not locate the data associated with this handle.
ACM_ACCESS_DENIED	The node secret does not exist.
ACE_TOO_MANY_CALLERS	An attempt was made to use a handle while a call was already in progress. This happens if a multithreaded application uses the same handle from multiple threads at the same time. This return code is not valid in this version.
ACE_INVALID_ARG	Input parameters are not valid.

Important: There is a design constraint on the use of SDI_HANDLE. An asynchronous application of the API must avoid concurrent use of the same SDI_HANDLE. Multiple threads using the same SDI_HANDLE return incorrect results.

Error Handling

To handle errors appropriately, use the value returned by this function at a decision point in your code. A successful return allows processing to continue.

Requirements

Minimum supported Authentication Manager Server Version	6.1 or later
Category	Synchronous API
Header	acexport.h
Static Library	UNIX: libaceclnt.a
Shared Library	Windows: • aceclnt.dll • aceclnt_tcp.dll • sdmsg.dll • xerces-c_3_1_vc80.dll UNIX: • libaceclnt.so • libaceclnt_tcp.so • libxerces-c-3.1.so AIX: • libaceclnt.so **Note:** Rename to libaceclnt.a. For more information, see “Using Dynamic Libraries or Static Libraries in UNIX” on page 38. • libaceclnt_tcp.so • libxerces-c-3.1.a

A Return Values and Result Codes

A return value indicates whether the function completed successfully. Result code refers to the setting of flags that you can check using the AceGetAuthenticationStatus function.

Return Values

The following table lists all the return values.

Return Value	Description
ACE_CFGFILE_NOT_FOUND	The **sdconf.rec** file could not be opened.
ACE_CFGFILE_READ_FAIL	The **sdconf.rec** file was too short or unreadable.
ACE_ERR_INVALID_HANDLE	The handle passed was invalid.
ACE_ERR_NOT_INITIALIZED	A call was made to an API function before AceInitialize was called.
ACE_INVALID_ARG	An invalid argument was supplied to an API function.
ACE_NOT_ENOUGH_STORAGE	The supplied buffer has insufficient storage for the data.
ACE_PROCESSING	This value is returned only by the asynchronous calls. The call was successful in initiating the requested asynchronous operation.
ACE_PTHREADATTR_FAIL	The UNIX pthread library failed to set a thread attribute. This return value is unusual. A systemic failure can cause this result. This value is not supported in this release.
ACE_PTHREAD_CREATE_FAIL	The UNIX pthread library failed to create the thread that processes the asynchronous requests. This return value is unusual. A systemic failure can cause this result. This value is not supported in this release.

Return Value	Description
ACE_SOCKET_LIB_NOT_FOUND	The Windows socket library winsock.dll could not be initialized or is the wrong version. This value will only be returned if you integrate with both SDK 8.5 and 8.1.
ACE_SUCCESS	The call was successful.
ACE_THREAD_CREATE_FAIL	The internal threads that control the asynchronous operations could not be created. This return value is unusual. A systemic failure can cause this result. This value will only be returned if you integrate with both SDK 8.5 and 8.1.
ACE_TOO_MANY_CALLERS	An attempt was made to use a handle while a call was already in progress. This happens if a multithreaded application uses the same handle from multiple threads at the same time. This return code is not valid in this version.
ACE_UNDEFINED_CLIENTADDR	The client IP address has not been set by ClientAddress.
ACE_UNDEFINED_NEXT_PASSCODE	The next tokencode has not been set by AceSetNextPasscode.
ACE_UNDEFINED_PASSCODE	The passcode has not been set by AceSetPasscode.
ACE_UNDEFINED_PIN	The PIN has not been set by AceSetPin.
ACE_UNDEFINED_USERNAME	The user name has not been set by AceSetUsername.

Result Codes

This table lists all the result codes.

Result Code	Description
ACM_ACCESS_DENIED	RSA Authentication Manager denied authentication. That is, the user's attempt to authenticate failed.
ACM_ACK_NAMELOCK	Not returned—for internal use only.
ACE_CHECK_PIN_REQ_NOT_KNOWN	RSA Authentication Manager PIN parameter user_selectable was not in the allowed range of values.
ACM_DOWNGRADE	Not returned—for internal use only.
ACM_EAP_INVALID_PEPPER	An EAP authentication request failed due to incorrect pepper value.
ACM_INVALID_SERVER	Used only for RSA ACE/Server 1.0. The Server node secret does not match the node secret of the Authentication Agent.
ACM_LOG_ACK	Not returned—for internal use only.
ACM_NEW_PIN_ACCEPTED	RSA Authentication Manager accepted the new PIN.
ACM_NEW_PIN_REJECTED	RSA Authentication Manager rejected the new PIN.
ACM_NEW_PIN_REQUIRED	The authentication was successful, but the user must select a new PIN.
ACM_NEXT_CODE_BAD	Not returned—for internal use only.
ACM_NEXT_CODE_REQUIRED	The authentication was successful, but the user must enter the next tokencode.
ACM_NO_SERVER	RSA Authentication Manager is not responding.
ACM_OK	The meaning of this result code depends on the function that caused it to be set. For more information, see "ACM_OK."
ACM_OK_2	Not returned—for internal use only.
ACM_OK_5	Not returned—for internal use only.
ACM_SHELL_BAD	Not returned—for internal use only.
ACM_SHELL_OK	Not returned—for internal use only.

ACM_SUSPECT_ACK	Not returned—for internal use only.
ACM_TIME_OK	Not returned—for internal use only.

ACM_OK

This table explains the meaning of the result code ACM_OK as it applies to individual function calls.

Function Call	**Description of ACM_OK**
SD_Init	The call was successful and the required resources were allocated.
AceCheck AceClientCheck SD_ClientCheckCredential AceSendNextPasscode SD_Check SD_CheckCredential SD_ClientCheck SD_Next SD_NextCredential SD_PinCredential	The authentication was successful.
AceClose SD_Close	The call was successful and resources were deallocated.
AceStartAuth AceContinueAuth AceCloseAuth	The call was successful, but no information is yet available on the authentication status.
AceLock SD_Lock	The call was successful.
SD_InitEx	RSA Authentication Agent is available, or if not, the offline authentication service is available.

B Using Synchronous and Asynchronous API Functions

Note: In SDK 8.5, all asynchronous calls from SDK 8.1 will be implemented as synchronous calls. The information in this appendix is no longer complete and will be updated in a later release of this guide.

Although the API functions share code, internal state resources exist that are managed by both synchronous and asynchronous calls. Mixing synchronous and asynchronous function calls produces unexpected results.

The following table shows which API function calls can be used with one another.

Function Call Group	Other Compatible Function Calls
The handles of the synchronous calls AceCloseAuth, AceContinueAuth, and AceStartAuth	Can be used with the synchronous calls AceGetAuthenticationStatus, AceGetShell, AceGetTime, and AceGetUserData. Can be used with the synchronous calls AceSetTimeout and Data. (The sample code file **\<AuthSDK-kit>\samples\sync** demonstrates the AceCloseAuth, AceContinueAuth, AceGetAuthenticationStatus, and AceStartAuth calls.)
The synchronous calls AceGetAlphanumeric, AceGetAuthAttribute, AceGetAuthenticationStatus, AceGetDAAuthData, AceGetDAuthenticationStatus, AceGetIterCountPolicy, AceGetLoginPW, AceGetMaxPinLen, AceGetMinPinLen, AceGetPepperPolicy, AceGetPinParams, AceGetShell, AceGetRealmID, AceGetSystemPin, AceGetTime, AceGetUserData, AceGetUserSelectable	Can be used with the synchronous calls SD_Check, SD_ClientCheck, SD_Close, SD_Init, SD_Next, and SD_Pin. Can be used with the asynchronous calls AceCancelPin, AceCheck, AceClientCheck, AceClose, AceInit, AceLock, AceSendNextPasscode, and AceSendPin.

Function Call Group	Other Compatible Function Calls
The synchronous calls AceSetAuthAttr, AceSetCredential, AceSetNextCredential, AceSetNextPasscode, AceSetPasscode, AceSetPin, AceSetPinCredential, AceSetTimeout, AceSetUserData, AceSetUsername	Can be used with the asynchronous calls AceCancelPin, AceCheck, AceClientCheck, AceClose, AceInit, AceLock, AceSendNextPasscode, and AceSendPin. **Note:** The AceSetTimeout and AceSetUserData functions can also be used with AceCloseAuth, AceContinueAuth, and AceStartAuth, as well as with the SD_xxx functions. (For examples of the synchronous calls AceSetNextPasscode, AceSetPasscode, AceSetPin, and AcesetUsername, see the sample code file **\<AuthSDK-kit>\samples\async**.)
The handles of the synchronous calls SD_Check, SD_ClientCheck, SD_Close, SD_Init, SD_InitEx, SD_Next, and SD_Pin	Can be used with the synchronous calls AceGetAlphanumeric, AceGetAuthenticationStatus, AceGetMaxPinLen, AceGetMinPinLen, AceGetPinParams, AceGetShell, AceGetSystemPin, AceGetTime, AceGetUserData, and AceGetUserSelectable. Can be used with the synchronous calls AceSetTimeout and Data. (The sample code file **\<AuthSDK-kit>\samples\sync2** demonstrates the SD_Check, SD_Close, SD_Init, SD_Next, and SD_Pin calls.)
The asynchronous calls AceCancelPin, AceCheck, AceClientCheck, AceClose, AceInit, AceSendNextPasscode, and AceSendPin	Can be used with the synchronous calls AceGetAlphanumeric, AceGetAuthenticationStatus, AceGetMaxPinLen, AceGetMinPinLen, AceGetPinParams, AceGetShell, AceGetSystemPin, AceGetTime, AceGetUserData, and AceGetUserSelectable. Can be used with the synchronous calls AceSetNextPasscode, AceSetPasscode, AceSetPin, and AcesetUsername. (The sample code file **\<AuthSDK-kit>\samples\async** demonstrates the AceCancelPin, AceCheck, AceClientCheck, AceClose, AceInit, AceSendNextPasscode, and AceSendPin calls.)

Call the function AceInitialize once before calling any other function. AceCleanup and AceShutdown work with any other function, but they operate only when needed to cancel authentications, close sockets, and terminate processing. Your code can make unlimited calls to AceCleanup. Make a call to AceShutdown only when you want to terminate all processing.

Checking the Status of Asynchronous Functions

All asynchronous calls return with a status that indicates success or failure of the function call, and not whether the user successfully authenticated. You must supply your own callback functions to determine the status of the actions initiated by the call.

As part of the callback method, your code can call AceGetAuthenticationStatus to find out the status of a specific authentication request. For more information, see "AceGetAuthAttr" on page 68.

Note: Before a successful return from an asynchronous function occurs, other threads are enabled. A developer-supplied callback function is called when these threads finish. You can determine the status of an authentication request only after all the threads associated with an asynchronous authentication call have successfully completed.

In addition, an asynchronous application of the API must avoid concurrent use of the same SDI_HANDLE. Multiple threads using the same SDI_HANDLE return incorrect results.

C Modifying the Message Catalog

The API functions AceStartAuth and AceContinueAuth return a set of prompts that the calling application must display to the user. The strings for these prompts are stored in a message catalog that you can customize. The message catalog also contains all of the messages that the API writes to the UNIX syslog and Windows Event Log.

To customize the strings on UNIX, use the UNIX gencat facility. To modify the strings on Windows, use the Microsoft Message Compiler, which is part of Microsoft Developer Studio.

Note: In the customizable message catalog for prompts and log messages, you must keep the size of prompts or log messages shorter than 1024 characters. Otherwise, the agent will fail. The prompts are further limited by the size of the developer-supplied promptStr character array used in the AceStartAuth and AceContinueAuth function calls.

Customizing Message Strings in UNIX

To customize the message strings:

1. Open **<AuthSDK-kit>/src/sdmsg.msg**, and modify the text.

   Note: You cannot add additional messages to this file.

2. Compile **sdmsg.msg** into **sdmsgcat.cat** by typing:

   ```
   gencat sdmsgcat.cat sdmsg.msg
   ```

3. Store the **sdmsgcat.cat** in the same directory as the **sdconf.rec**.

Customizing Message Strings in Windows

To customize the message strings:

1. Compile **<AuthSDK-kit>\src\sdmsg.mc** with Microsoft Message Compiler using the following command:

   ```
   mc sdmsg.mc
   ```

   This compilation creates **MSG00001.BIN**.

2. Make a backup of **SDMSG.DLL** in your installation. Open the original **SDMSG.DLL** with Microsoft Developer Studio using the **Open as Resources** option.

3. Open **MSG00001.BIN** with Microsoft Developer Studio and copy the entire contents shown as binary data to the clipboard.
4. In **SDMSG.DLL**, open the resource of type 11 with ID 1, and select the entire content.
5. Paste the contents from the Clipboard into **SDMSG.DLL**.
6. Modify the version resource description information of **SDMSG.DLL** to indicate that it is a customized file.
7. Save **SDMSG.DLL**.

D Deployment Guidelines

Install, upgrade and uninstall options for an agent may differ depending on the deployment environment. This section describes guidelines and important points to be considered during installation and upgrade in different deployment scenarios.

Single Agent on a Host

A single agent on a host machine may store the libraries (**aceclnt.dll** or **aceInit.so**) and configuration files (**sdopts.rec**, **sdconf.rec**), either in the default location or in a customized location.

The agent can be upgraded to the 8.5 APIs by updating the binaries to the latest version. See "Upgrading from 8.1" on page 29.

Note: SDK 8.5 does not create **sdstatus12** and **securid** files. The new Agent creates additional files: **bootstrap.xml**, **config.xml,** and **root.cer**.

Multiple Agents on a Host Authenticating with a Common Authentication Manager

Multiple agents on the same host, authenticating with the same Authentication Manager, may store the configuration files and binaries in the default location or in a customized location. That is, they can either use common configuration files and binaries or have separate files.

The possible deployment types:

Common Configuration Files and Common Binaries

Common Configuration Files and Separate Binaries

Separate Configuration Files and Common Binaries

Separate Configuration Files and Separate Binaries

Common Configuration Files and Common Binaries

Install Scenario

An Agent-A using the 8.5 API is installed on a host and successfully authenticating with the Authentication Manager server. The configuration files and **aceclnt** binaries are stored in the default location.

An Agent-B using the 8.5 API is installed on the same host machine. Agent-B can load the **aceclnt** binaries from the default location and use the configuration files created by Agent-A, and start communicating with the Authentication Manager server.

Upgrade

When upgrading to 8.5 API, both agents must be upgraded. Since the binaries used are the same, when one agent is upgraded, the other agent automatically gets upgraded. For upgrade instructions, see "Upgrading from 8.1" on page 29.

Uninstall

If one of the agents is uninstalled, shared configuration files and **aceclnt** binaries used by the other agent may get deleted, thus breaking the other agent.

Common Configuration Files and Separate Binaries

Install Scenario

An Agent-A using the 8.5 API is installed on a host and successfully authenticating with the Authentication Manager server. The **aceclnt** binaries and configuration files are stored in the default location.

An Agent-B using the 8.5 API is installed on the same host machine. Binaries are stored in a customized location. Configuration files are stored in the default location. Agent-B can use the configuration files created by Agent-A and start authenticating with the Authentication Manager server.

Upgrade

Note: The node secret is optional in SDK 8.5. For information about deploying an IPv4/IPv6 authentication agent, see the *RSA Authentication Manager 8.x Adminstrator's Guide*.

Uninstall

If one of the agents is uninstalled, the shared configuration files used by the other agent may get deleted, thus breaking the other agent.

Separate Configuration Files and Common Binaries

Install Scenario

An Agent-A using the 8.5 API is installed on a host and successfully authenticating with the Authentication Manager server. The configuration files and **aceclnt** binaries are stored in the default location.

Note: The node secret is optional in SDK 8.5. For information about deploying an IPv4/IPv6 authentication agent, see the *RSA Authentication Manager 8.x Adminstrator's Guide*.

An Agent-B using the 8.5 API is installed on the same host machine, loads the **aceclnt** binaries from the default location and stores the configuration files in a customized location. Since the server has already created a node secret for the host, the node secret and other configuration files of Agent-A must be copied to the customized location of Agent-B, before Agent-B can communicate with the Authentication Manager server.

Upgrade

When upgrading to 8.5 AP1, since the binaries are common, both agents must be upgraded. For upgrade instructions, see "Upgrading from 8.1" on page 29.

Uninstall

If one of the agents is uninstalled, shared **aceclnt** binaries used by the other agent may get deleted, thus breaking the other agent.

Separate Configuration Files and Separate Binaries

Install Scenario

An Agent-A using the 8.5 API is installed on a host and successfully authenticating with the Authentication Manager server. The configuration files and **aceclnt** binaries are stored in the default location.

An Agent-B using 8.5 API is installed on the same host machine. Configuration files and **aceclnt** binaries are stored in customized locations.

Note: The node secret is optional in SDK 8.5. For information about deploying an IPv4/IPv6 authentication agent, see the *RSA Authentication Manager 8.x Adminstrator's Guide*.

Since the server has already created a node secret for the host, the node secret and other configuration files of Agent-A must be copied to the customized location of Agent-B, before Agent-B can communicate with the Authentication Manager server.

Upgrade

Since the binaries and configuration files for each agent are separate, you can upgrade one agent at a time. For upgrade instructions, see "Upgrading from 8.1" on page 29.

The node secret is optional in SDK 8.5. For information about deploying an IPv4/IPv6 authentication agent, see the *RSA Authentication Manager 8.x Adminstrator's Guide*.

Uninstall

Since the configuration files and binaries for each agent are stored separately, one agent can be uninstalled without affecting the other.