



RSA ClearTrust 4.7

Developer's Guide

Contact Information

See our Web sites for regional Customer Support telephone and fax numbers.

RSA Security, Inc.
www.rsasecurity.com

RSA Security Ireland Limited
www.rsasecurity.ie

Trademarks

ACE/Agent, ACE/Server, BSAFE, ClearTrust, JSAFE, Keon, RC2, RC4, RC5, RSA, SecurCare, SecurID, SoftID and WebID are registered trademarks, and BCERT, Because Knowledge is Security, RC6, RSA Security, RSA Secured, SecurWorld, The Most Trusted Name in e-Security, the RSA logo and the RSA Secured logo are trademarks of RSA Security Inc.

Other product and company names mentioned herein may be the trademarks of their respective owners.

License agreement

This software and the associated documentation are proprietary and confidential to RSA Security, 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 below. This software and any copies thereof may not be provided or otherwise made available to any other person.

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 exporting this product.

Distribution

Limit distribution of this document to trusted personnel.

© RSA Security 2002. All Rights Reserved.

First Printing: March 2002

P/N 3503A0

Contents

Preface	viii
About This Guide.....	viii
Related Documentation.....	ix
Document Conventions.....	ix
Typographical Conventions.....	x
Comment Icons.....	xi
Getting Support and Service.....	xi
Chapter 1: Overview of the RSA ClearTrust APIs	1
The RSA ClearTrust APIs.....	1
The Administrative API.....	2
The Runtime API.....	4
The WAX API.....	4
Coding Recommendations.....	4
Multithreaded Programming.....	5
Using the RSA ClearTrust API Efficiently.....	5
Chapter 2: Installing the RSA ClearTrust APIs	7
Installing APIs on Windows.....	7
Installing APIs on Solaris.....	9
Chapter 3: Administrative C API	11
This Chapter.....	11
Installing and Compiling.....	12
Location.....	12
Sample Code.....	12
Header Files.....	12
API Libraries.....	13
Building for UNIX.....	14
Building for Windows 2000 and NT.....	14
Initialization and Login Operations.....	15
Initialization.....	15
Login.....	16
Connecting With and Without SSL.....	17
The Functions of ct_commands.h.....	18
Functions For Loading Objects.....	18
Administrative Functions.....	18
Password Setting Functions.....	19
Deprecated Runtime-Type Functions.....	19
Administrative Objects.....	20
Administrative Group (VBU).....	20

Administrative User.....	24
Administrative Role.....	25
Password Policy	27
Participants.....	28
CT_EntityHdr Struct	28
Groups.....	29
Users	31
User Properties	33
User Property Definitions.....	33
Deprecated Structure: Realms.....	35
Policy Objects.....	36
Basic Entitlements	36
SmartRules.....	40
Resources.....	42
Applications.....	42
Application Functions	44
Application URLs.....	44
Web Servers.....	45
Server Trees.....	46
Searching.....	47
Permissions	52
Object Utilities.....	54
Error Codes	54
Memory Management in the C API	56
Memory Management when Getting API Objects.....	56
Memory Management when Modifying an API Object	57
Memory Management when Creating API Objects.....	57
Sample Code	60
AdminUser.c.....	60
Chapter 4: Administrative Java API	67
This Chapter.....	67
Installing and Compiling.....	68
Compiling Applications	68
APIServerProxy.....	69
APIServerProxy Method Reference.....	69
Connecting an APIServerProxy Client.....	72
Disconnecting an APIServerProxy Client.....	74
Connection Example	74
Administration Objects	78
Administrative Group.....	78
Administrative User.....	80

Administrative Role.....	82
Password Policy	85
Participants.....	87
Groups.....	87
Users	89
User Properties	91
User Property Definitions	91
Deprecated Interface: IRealm	93
Policy Objects.....	94
Basic Entitlements (Explicit Entitlements).....	94
SmartRules.....	95
Resources.....	96
Applications.....	96
Application Functions	97
Application URLs.....	99
Web Servers.....	99
Server Trees.....	101
Utility Classes.....	102
ISparseData.....	102
Permissions	103
Criteria.....	105
Boolean Criterion.....	105
Date Criterion	105
Float Criterion	106
Integer Criterion.....	106
Searching.....	106
Administrative Group Search.....	107
Application Search.....	107
Group Search	107
Deprecated: Realm Search.....	108
User Property Definition Search.....	108
Web Server Search.....	108
User Search	109
Examples	110
User Example	110
User Property Example	113
Application Function Example	120
SmartRule Example.....	123
User Search Example	127

Chapter 5: Runtime C API	131
This Chapter	131
C Runtime API Overview	131
Authentication	132
Authorization	132
SSO Token Manipulation	133
User Property Retrieval	133
Installing and Compiling	134
Location	134
Header Files	134
Connecting a Runtime C API Client	135
SSL and Non-SSL Connection Options	136
Access to Tokens and User Properties	136
Connection Pool Functions and Keys	138
Runtime C API Reference	141
Client Keys	141
Authentication Types	145
Runtime Functions	149
Maps	151
Examples	153
RSA SecurID Authentication Example	153
Chapter 6: Runtime Java API	157
This Chapter	157
Overview	158
What the Runtime API Does	158
Runtime API Relies on Authorization Servers	160
Runtime API Calls Are Threadsafe	160
Runtime API vs. Administrative API	160
Installing and Compiling	161
Compiling Applications	161
Client Connection Options	162
Access to Tokens and User Properties	162
Connecting Over Authenticated SSL	164
Connecting Over Anonymous SSL	166
Connecting Without SSL	166
Packages	167
Interfaces	167
Interface RuntimeAPI	167
Interface UserConstants	169
Interface TokenKeys	169
Interface AuthTypes	169

Interface ResourceConstants	170
Interface ResultConstants	170
Interface CredentialConstants	170
Runtime API Classes	171
Class APIFactory	171
Class ServerDescriptor	171
Examples	172
Runtime API Example Without SSL	172
Runtime API Example With SSL	177
RSA SecurID Authentication Example	183
Chapter 7: Administrative and Runtime DCOM API	189
Requirements	189
Installing the DCOM API	190
Using the DCOM API	194
Instantiating and Connecting	194
Getting Objects	195
Making RSA ClearTrust API Calls	195
Classes in the sirrus.api.com Package	196
SecurantDCOMFactory	196
AuthTypesClass	196
ResourceConstantsClass	196
ResultConstantsClass	197
UserConstantsClass	197
UserPropertyTypesClass	197
DCOM API Example Code	198
DCOM Runtime API example	198
ASP page, create user	200
ASP page, get users list	201
Chapter 8: Web Agent Extension API	203
Overview	203
Extending the Web Server Agent	204
How an Agent Processes a URI Request	205
Agent Phase Handlers	207
Path Check Handler	207
Session Handler	207
Pre-Authentication Handler	208
Authentication Handler	208
Authorization Handler	209
Cookie Handler	210

Writing a WAX Program	212
Overview	212
WAX API Headers	212
WAX API Libraries	213
Registering a WAX Program	213
Writing a WAX Method	213
Registering a WAX Method	214
Invoking a WAX Authentication Method	215
Compiling and Linking a WAX Program	215
WAX Examples	218
Cookie Data Example	218
Custom Authentication Example	221
Custom Error Pages Example	224
WAX API Reference	227
The ct_wax_init Initialization Method	227
ct_extension_init	228
Hash Table Functions	228
Memory Management	228
Printing Status and Debug Information	229
Request Data	229
Status Handler	231
Loading Parameter Settings	233
Using WAX Programs with Virtual Host-Enabled Servers	233
Chapter 9: Customizing Your Web Environment	235
Personalizing the Environment	235
Creating Personalized Content	236
RSA ClearTrust Environment Variables	236
Details	237
Contents of the RSA ClearTrust Cookie	239
Changing the Cookie Name	239
Writing ASP and JSP Pages	240
RSA ClearTrust Parameter Names	240
Password Changer Example	240
HTTP Header Parameters	253
Index	255

Preface

This *Developer's Guide* provides a complete overview of the RSA ClearTrust® application programming interfaces (APIs). This guide explains the RSA ClearTrust API libraries, and provides references and usage examples.

About This Guide

This guide describes the RSA ClearTrust application programming interfaces (APIs). The intended audience of this document is Java and C programmers, Web developers, or systems engineers responsible for developing custom software applications that interact with the RSA ClearTrust system. This guide assumes that you are proficient in either the C or Java programming language.

This guide contains the following chapters:

- [Chapter 1, “Overview of the RSA ClearTrust APIs”](#). This chapter provides diagrams and outlines of the APIs.
- [Chapter 2, “Installing the RSA ClearTrust APIs”](#). This chapter shows how to install the APIs from the RSA ClearTrust product CD.
- [Chapter 3, “Administrative C API”](#). This chapter describes the C version of the RSA ClearTrust Administrative API, showing how to develop security administrator applications that create/update user accounts and set the access rules enforced by the RSA ClearTrust system.
- [Chapter 4, “Administrative Java API”](#). This chapter describes the Java version of the RSA ClearTrust Administrative API, showing how to develop security administrator applications in Java.
- [Chapter 5, “Runtime C API”](#). This chapter explains how to build client applications in C that can perform authentication, authorization, and other functions using the runtime functionality of the RSA ClearTrust Authorization Servers.
- [Chapter 6, “Runtime Java API”](#). This chapter explains how to build client applications in Java that can perform authentication, authorization, and other functions using the runtime functionality of the RSA ClearTrust Authorization Servers.
- [Chapter 7, “Administrative and Runtime DCOM API”](#). This chapter explains the RSA ClearTrust DCOM API. This API allows ASP pages to use the administrative and runtime features of the RSA ClearTrust API. The RSA ClearTrust DCOM API is implemented by accessing the RSA ClearTrust Java API via a bridge layer.
- [Chapter 8, “Web Agent Extension API”](#). This chapter explains how to extend and customize the functionality of the RSA ClearTrust Web Server Agents. An extension you write using this API is called a Web Agent Extension (or “WAX” for short).

- **Chapter 9, “Customizing Your Web Environment”.** This chapter provides information about various customizations and personalizations that you can implement in your RSA ClearTrust-protected Web servers. Many of these are provided as samples to get you started developing and creating your own customized and personalized forms and Web server applications.

Related Documentation

For more information about the RSA ClearTrust product, refer to the following guides in this 4.7 documentation set:

- **Overview Guide.** This guide provides a comprehensive overview of the system components, supported platforms, and features of RSA ClearTrust.
- **Installation and Configuration Guide.** This guide provides instructions for installing and configuring the RSA ClearTrust Servers, Data Adapters, Web Server Agents and the Entitlements Manager Web-based administration tool on your chosen operating system. This guide also contains detailed descriptions of the different configuration options, features and production environment considerations.
- **Administrator's Guide.** This guide provides information for your Security Administrators on how to administer users and security policy in RSA ClearTrust. There are instructions for administering users, resources and security policy in the *RSA ClearTrust Entitlements Manager* Web-based administration tool. For more information about using the *Entitlements Manager*, you can also refer to the online help files.

Document Conventions

These document conventions are used consistently throughout RSA ClearTrust's documentation to help you identify certain types of information.



Typographical Conventions

Convention	Meaning	Example
San-serif bold	User interface elements such as buttons, menus choices, window names, dialog boxes, field names and so on will appear in san-serif bold text.	Select File ▶ Print . Click Save .
SAN-SERIF BOLD UPPERCASE	Keyboard keys, including letters and numbers as well as Tab, CTRL, ALT, and so on, will appear in san-serif bold uppercase text.	Press CTRL+ALT+DELETE
<i>Italics</i>	New terms, emphasized words or book titles will appear in italics.	See the <i>Administration Guide</i> for more information about using the Entitlements Manager.
UPPERCASE	Environment variables, SQL commands, logical operators, device names, acronyms, registry settings, system commands, and so on will appear in all uppercase letters.	SELECT object_name FROM user_objects Mount your CD-ROM drive.
Courier	Code examples, files, directories, class names, commands, parameters and on-screen computer output will appear in courier font.	Edit the <code>aserver.conf</code> file in the <code>\conf</code> directory.
Courier bold	Typed input, as opposed to on-screen computer output, will appear in bold courier font.	Enter the hostname of your Web server here: web1.rsa.com
<i><italics></i>	Italicized text contained within the less than (<) and greater than (>) symbols denotes information to be determined by the reader. Substitute the appropriate name, directory, or other specific information.	<code>print <filename></code> <code><ct_home>/cleartrust/conf</code>
[]	Text contained within square brackets denotes optional information.	<code>reject [-d] <filename></code>
	Text separated by the pipe symbol denotes an either/or relationship.	<code>true false</code>
\$	Bourne, Bourne Again or Korn shell prompt for UNIX commands	\$

Convention	Meaning	Example
%	C shell prompt for UNIX commands	%
#	Super user or root prompt for UNIX commands	#

Comment Icons

Comment icons identify particular types of information, as the following table describes.

Icon	Alert Labels	Description
	Warning: Important:	Identifies paragraphs that contain vital instructions, cautions or critical information.
	Note: Tip:	Identifies paragraphs that contain notes, RSA recommendations or other helpful product information.

Getting Support and Service

SecurCare® Online	www.rsasecurity.com/support/securcare
General Technical Support Information	www.rsasecurity.com/support

1

Overview of the RSA ClearTrust APIs

The RSA ClearTrust APIs

The APIs give you programmatic access to the RSA ClearTrust Servers at all levels. There are three sets of API libraries available with RSA ClearTrust:

- The **Administrative API** allows you to develop security administrator applications that create/update user accounts and set the access rules enforced by the RSA ClearTrust software. Administrative API calls will perform read/write operations on the Entitlements Database via the Entitlements Server (called the “API Server” when accessed by an API client). There are Java, C, and DCOM versions of the Administrative API.
- The **Runtime API** allows you to use the RSA ClearTrust system to provide user authentication and authorization for applications or services that are not covered by the RSA ClearTrust system out of the box. For example, if you have a custom-built application that needs to control user access to itself, it can use the Runtime API to check users’ permissions in the RSA ClearTrust system. The Runtime API has read-only access to the Entitlements Database. There are Java, C, and DCOM versions of the Runtime API. The Administrative and Runtime APIs maybe used together in a single client program, if needed.
- The **Web Agent Extensions (WAX)** allow you to augment or override steps in the RSA ClearTrust authentication and authorization process. By adding WAXes, you can, for example, check a custom database for a user’s authorization status. WAXes must be written in C.

Additional interface points allow you to customize your RSA ClearTrust Web environment.

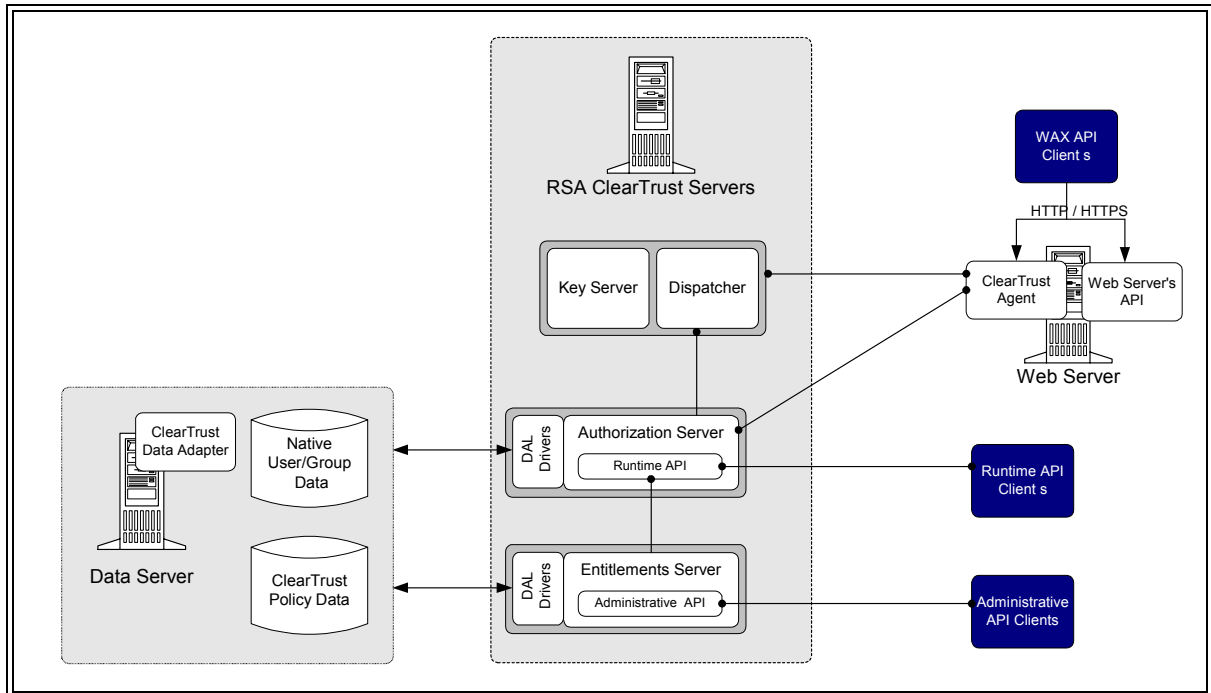


Figure 1.1 RSA ClearTrust APIs and System Architecture

The Administrative API

This section provides an overview of the Administrative API objects and describes the common elements of both platforms (C and Java).

The RSA ClearTrust Administrative API enables application developers to implement custom functionality—typically in the area of security policy administration such as batch importing of users from a proprietary database—into the back-end RSA ClearTrust Servers.

The primary advantage of the RSA ClearTrust system as a security system is in its flexibility and extensibility. The RSA ClearTrust system provides a powerful turn-key solution for securing proprietary applications on Web Servers.

To facilitate the development of these applications, RSA Security provides versions of the RSA ClearTrust APIs for Java, C, and DCOM. C++ applications can use the C API, and Visual C++ applications also have the option of using the object oriented DCOM API.

The following figure illustrates the object model of the RSA ClearTrust API and the relationships between the objects.

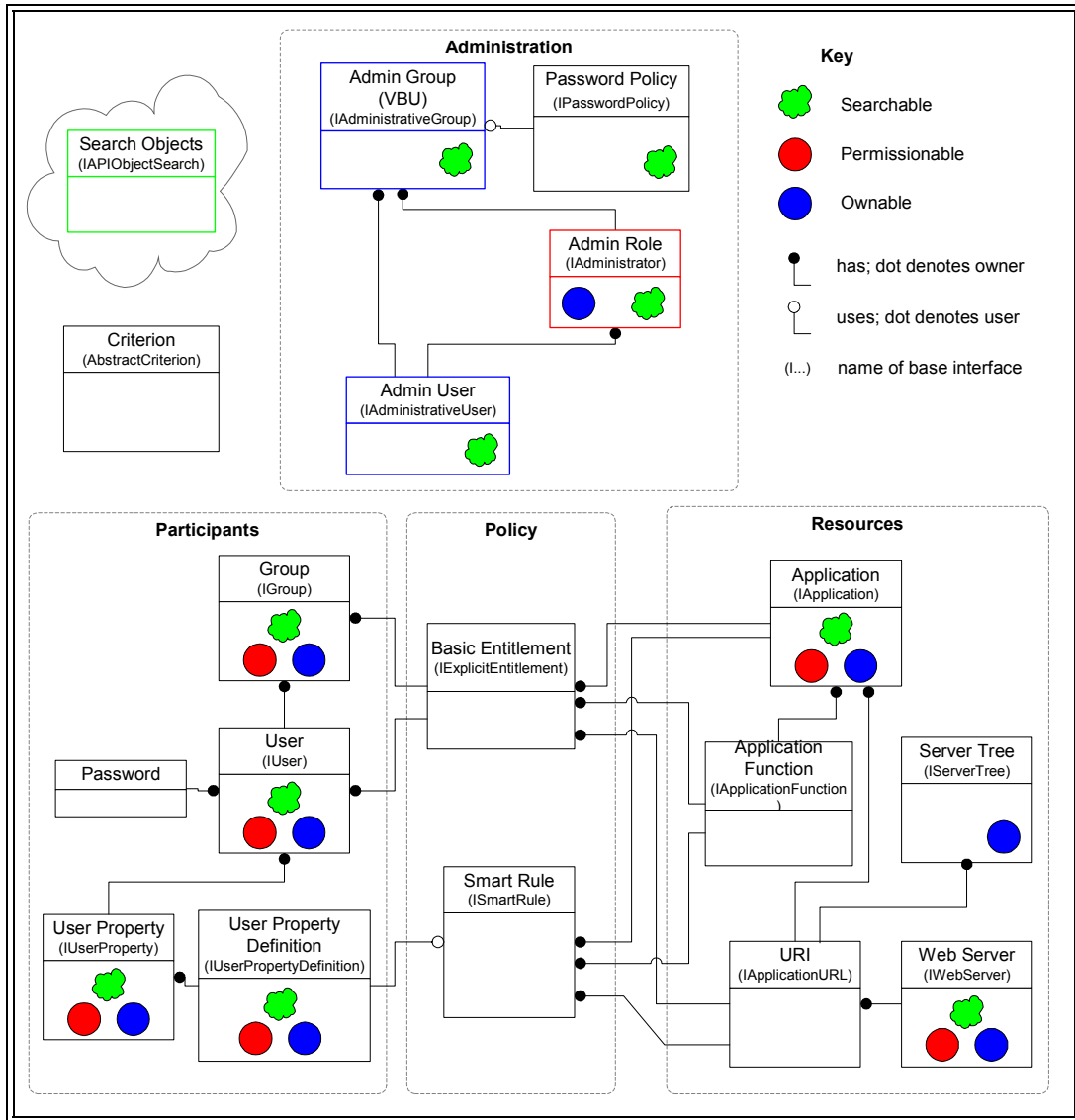


Figure 1.2 RSA ClearTrust Object Model

The Runtime API

The RSA ClearTrust Runtime API enables trusted client applications to gain full access to the runtime functionality of the RSA ClearTrust Authorization Servers. This functionality consists of two main pieces: authentication of users, and authorization of users to resources. A defining characteristic of both of these tasks is that they are read-only; that is, they perform queries on the existing state of the data in the RSA ClearTrust Servers, but do not involve changing or adding data. Both the C and Java versions of the Runtime API provide access to this runtime functionality in the same highly available manner as is enjoyed by the RSA ClearTrust Web Server Agents, incorporating internal connection pools and failover logic.

The WAX API

The RSA ClearTrust® Web Server Agents provide the RSA ClearTrust Web Agent Extension API (the “WAX API”), which allows developers to extend and customize the functionality of any RSA ClearTrust Agent. For example, your extensions may perform custom authentication or manipulate dynamic, user-specific content.

Unlike the RSA ClearTrust Administrative API, the RSA ClearTrust Web Agent Extension API does not modify the RSA ClearTrust database, rather it controls the behavior of the RSA ClearTrust Agent during the authentication and authorization processing. An extension you write using this API is called a Web Agent Extension (or “WAX” for short).

For example, you could extend the functionality of the RSA ClearTrust Web Server Agent in the following ways.

- Create an extension directing the Web server to a custom HTML file for different return codes from the RSA ClearTrust Authorization Server.
- Create an extension providing custom logging.
- Create an extension providing custom authentication of users.
- Create an extension integrating the RSA ClearTrust Agent with proprietary Web Server Agents or third-party Agents.

Coding Recommendations

RSA ClearTrust recommends certain coding practices for optimal use of the APIs. This section presents recommendations for supporting multi-threaded programs and for using the APIs efficiently.

Multithreaded Programming

The RSA ClearTrust APIs support connections from multithreaded programs, such as Microsoft Transaction Server.

C API

The C API supports two sets of functions. One set operates on a single connection, the other provides a context that can be used to operate on multiple connections. Both sets of functions are thread-safe, however, using the context functions for multiple connections provides for increased flexibility and performance.

The context-enabled functions take all the same arguments as the non-context-enabled versions, except that each context-enabled function (denoted by the “ctxt_” in its function name) additionally takes `ct_context` as its first argument, to enable the RSA ClearTrust API Server to track the client connections. (The original C API uses a default context that will not conflict with any of the contexts that are exposed as `ct_contexts`.) For example, there are two functions for saving an administrative group: `ct_save_admin_group()` and `ctxt_save_admin_group()`.

Java API

The Java API does not support the notion of a context. Multiple instances of the API can be run against the server proxy.

Using the RSA ClearTrust API Efficiently

When you are developing an application with the RSA ClearTrust API, you can do the following to increase performance:

- connect only once
- minimize the number of API calls
- use the user and user property functions

The following subsections describe these strategies in further detail.

Connect Only Once

The RSA ClearTrust API uses a persistent connection to the server; once you connect to the API Server you need not set up the connection again until the connection is lost. The API Server enforces administrative security by ensuring that an API Client cannot be used to change the RSA ClearTrust database maliciously. Because the connection involves some overhead cost, it is more efficient to connect just once rather than connecting every time you wish to modify the database.

Minimize the Number of API Calls

Every API call made involves network communication. To perform a sequence of API calls, first decide if the number of API calls can be reduced to just a few. For instance, to retrieve a group of objects, it is more efficient to use the `getByRange` or `getByNames` function than to retrieve each object individually.

Incorrect Usage Example

The following example shows inefficient code for displaying the names of a group of users. The problem with this code is that using the `size()` function within the FOR loop causes a remote command to occur on the server, which will be very slow. Also note that the `getByIndex(i)` call requires a network call to the server, further slowing down the program.

```
ISparseData theUsers = myServerProxy.getUsers();
for (i =0;i <theUsers.size();i++)
{
    // Each getByIndex call causes a
    // remote command on the server.
    IUser aUser = (IUser) theUsers.getByIndex(i);
    System.out.println("User["+ i + "] = "+aUser.getName());
}
```

Correct Usage Example

The following example shows the correct way to display the names of a group of users. This code acquires all users in just two function calls, so it is very efficient. An array is returned, and then the length of the array is used to drive the counter for the loop. (Always work off the length of the returned array; since it's possible to receive fewer objects than requested.) References to user data are then made to array, which is in local storage, so this is much faster than retrieving data from the server.

```
ISparseData theUsers = myServerProxy.getUsers();
int numofUsers = theUsers.size();
IAPIObject [] userArray =
    theUsers.getByRange(0, numofUsers-1);
for (i =0;i <userArray.length;i++)
{
    IUser aUser = (IUser) userArray[i];
    System.out.println("User[" + i + "] = " +
        aUser.getName());
}
```

Use the User and User Property Functions

To retrieve a user and all of its properties, use the `get_user_and_properties` function in C and the `getUserAndProperties` call in Java. This is more efficient than retrieving a user and each of its properties one at a time.

2

Installing the RSA ClearTrust APIs

This chapter provides instructions for installing the RSA ClearTrust APIs on Windows and on Solaris. See:

- “[Installing APIs on Windows](#)” below, or
- “[Installing APIs on Solaris](#)” on page 9.

Installing APIs on Windows

When installing the RSA ClearTrust APIs on Windows, an InstallShield® program will guide you through the installation.

1. Insert your installation CD into the CD-ROM drive and navigate to `\Windows\ct_servers`.
2. Double-click the **Setup.exe** icon to launch the InstallShield® program.
3. In the **Welcome** dialog, click **Next** to begin the installation.
4. In the **Select Region** dialog, select your geographical region and click **Next**.
5. In the **License Agreement** dialog, click **Yes** to accept the license agreement and continue with the installation.
6. In the **Choose Destination Location** dialog, select an installation location for the RSA ClearTrust software, and click **Next**. The default is `C:\Program Files\RSA\ClearTrust`.
7. In the **Setup Type** dialog, select the **Custom** type of installation and click **Next**.



Note: The API files are available only from the **Custom** installation option.

8. In the Select Components dialog, click the check boxes for the ClearTrust API components and any other components you wish to install. In most cases, you will install all components. If you need additional information on installation options, see the *RSA ClearTrust Installation and Configuration Guide*.

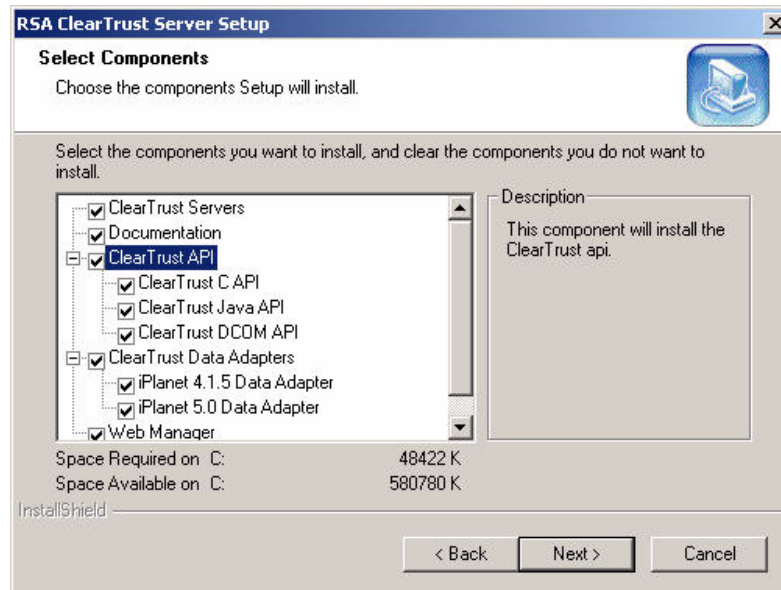


Figure 2.1 The Select Components dialog

9. If you are also installing the RSA ClearTrust servers, enter the following information in the **Dispatcher Configuration** dialog and click **Next** to configure your Dispatcher Server:
 - **RSA ClearTrust Administrator e-mail address.**
 - **Authorization Server list port.** Default is 5608
 - **Authorization Server registration port.** Default is 5607
10. If you are also installing the RSA ClearTrust servers, enter information about your primary LDAP directory server in the **LDAP Server Connection Information** dialog and click **Next**.
 - **LDAP Server Hostname:** This is the fully qualified hostname or IP address of your LDAP directory server machine. This writes to the `cleartrust.data.ldap.directory.<iplanet>.hostname` parameter in your `conf\ldap.conf` file.
 - **LDAP Server Port:** This is the default port number that RSA ClearTrust will use to make a connection to the LDAP directory. This writes to the `cleartrust.data.ldap.directory.<iplanet>.port` parameter in your `conf\ldap.conf` file. The default LDAP port is 389. The default LDAPS port is 636.
11. If you are also installing the RSA ClearTrust servers, enter the following information in the **LDAP Configuration** dialog and click **Next** to configure your LDAP Data Store:

- **LDAP Bind DN:** When connecting to the LDAP server, this is the bind DN (the directory manager logon name) that will be used. This writes to the `cleartrust.data.ldap.directory.<iplanet>.binddn` parameter in your `conf\ldap.conf` file.
 - **LDAP Password:** When connecting to the LDAP server, this is the password for the administrator's account listed above. This writes to the `cleartrust.data.ldap.directory.<iplanet>.password` parameter in your `conf\ldap.conf` file.
- 12.** The installation program displays a progress bar indicating the files currently being installed. When the process is complete, click **Finish** to complete the installation program.

After the RSA ClearTrust installation is finished, you may turn to the “Installation” section in any of the API chapters of this book for an explanation of the files and directories that make up the API.

Installing APIs on Solaris

When installing the RSA ClearTrust APIs, the installer will prompt you for environment variables and other required information before installing the packages. These instructions assume you are logged in on the console of the machine, running OpenWindows or CDE, and have an open shell (either via `cmdtool`, `xterm`, or CDE terminal).

1. Logon as user `root`.
2. Mount your CD-ROM drive and go to `/Solaris/api`
3. Install the RSA ClearTrust APIs package with a `pkgadd` command:

```
# pkgadd -d ./RSActapi-4.7-solaris-sparc.pkg
```

4. At the first prompt, type `all` to install all packages.
5. At the next prompt, accept the displayed license agreement.
6. Follow the prompts for directory information, entering information as required:
 - **Installation Base Directory.** In a typical installation you should accept the default value of `/opt`
 - **RSA ClearTrust package root.** Default is `/opt/ctrust/api`. RSA Security recommends accepting the default.

If the installation succeeds, it will display the following:

```
Installation of <RSActapi> was successful.
```

After the RSA ClearTrust installation is finished, you may turn to the “Installation” section in any of the API chapters of this book for an explanation of the files and directories that make up the API.

3

Administrative C API

This chapter describes the C version of the RSA ClearTrust Administrative API. The Administrative API allows you to develop security administrator applications that create/update user accounts and user groups and set the access policies enforced by the RSA ClearTrust system. The Administrative API uses the Entitlements Server (called the “API Server” when accessed by an API client) to write to the user, policy and administrator data stores on your configured LDAP directory server.

The C API contains two sets of functions: one set requires a context parameter while the other set does not. This chapter only describes the non-context version of the C functions.

This chapter contains brief overviews of each function. For more detailed descriptions, including the input/output parameters and return values of each function, see the API header files. You can find the header files in your RSA ClearTrust installation in the directory `<CT_HOME>/api/admin-c/include`.

This Chapter

This chapter consists of:

- Compilation instructions starting with [“Installing and Compiling”](#) on page 12.
- Instructions for connecting an Administrative API client, in [“Initialization and Login Operations”](#) on page 15.
- Reference information on the C API structures and methods:
 - [“The Functions of ct_commands.h”](#) on page 18.
 - [“Administrative Objects”](#) on page 20.
 - [“Participants”](#) on page 28.
 - [“Policy Objects”](#) on page 36.
 - [“Resources”](#) on page 42.
 - [“Searching”](#) on page 47.
 - [“Permissions”](#) on page 52.
 - [“Object Utilities”](#) on page 54.
 - [“Error Codes”](#) on page 54.
 - [“Memory Management in the C API”](#) on page 56.
- Example programs in [“Sample Code”](#) starting on page 60. The source code for these API example programs is installed in `<CT_HOME>/api/admin-c/example`

Installing and Compiling

This section explains the installed components that make up the API and provides guidelines for building applications. For instructions on installing the APIs, see [Chapter 2, “Installing the RSA ClearTrust APIs”](#).

Location

The files you need to build and run Administrative API applications can only be installed when the RSA ClearTrust Servers are installed. In order to install the API, make sure you choose the **Custom** install type and then select all of the API components to install them. See the *Installation and Configuration Guide* for details.

The C Administrative API files are located in `<CT_HOME>/api/admin-c`. In addition, you will need libraries located elsewhere in your RSA ClearTrust installation. See [“API Libraries”](#) on page 13 for details.

Sample Code

The code examples shown at the end of this chapter are provided in your installation under:

```
<CT_HOME>/api/admin-c/example
```

Header Files

API Headers

The following header files define the methods and types of the RSA ClearTrust Administrative API. These files are located in `<CT_HOME>/api/admin-c/include`.

- `ct_commands.h` is the main RSA ClearTrust Administrative API header file and contains all the API session initialization functions as well as commands for finding, loading and saving user, resource, entitlement, and SmartRule records.
- `ct_bool.h` defines the boolean type and platform-dependent datatypes.
- `ct_map.h` contains utilities for working with Maps. Maps are multivalued datatypes used in the RSA ClearTrust system to describe objects like users and authentication results.
- `ct_permissions.h` includes methods that check an administrative user's right to perform each type of administrative action in the RSA ClearTrust system.
- `ct_rc_constants.h` defines the RSA ClearTrust return codes and error codes.
- `ct_search.h` contains search methods for finding and loading administrative objects such as users, groups, entitlements and resources based on user-specified criteria.
- `ct_structs.h` defines the RSA ClearTrust administrative API object types.

- `ct_utilities.h` contains memory management functions.

Removed Headers

The following header files no longer exist. Instead, use the methods in `ct_commands.h`.

- `ct_objects.h`
- `ct_relations.h`

Headers Included for Compilation Only

The following header files are internal files included only because they are required when building API programs.

- `ct_auth_result.h`
- `ct_boolean.h`
- `ct_error.h`
- `ct_hash.h`
- `ct_lock.h`
- `ct_lock_impl.h`

API Libraries

The RSA ClearTrust C Administrative API is implemented as a C-to-Java JNI wrapper around the RSA ClearTrust Java Administrative API. For this reason, you must install both the RSA ClearTrust C API library and the RSA ClearTrust Java API library in order to build and run C Administrative API applications. In addition, some SSL libraries are required if you are connecting to the RSA ClearTrust API Server over SSL. The required libraries are

- the C Administrative API library, located in `<CT_HOME>/api/admin-c/lib`.
On **Solaris**, this will be either
 - static: `libct_admin_api.a`, or
 - dynamic: `libct_admin_api.so`On **Windows**, this will be either
 - static: `ct_admin_api.lib`, or
 - dynamic: `ct_admin_api.dll`
- the Java Administrative API library:
`<CT_HOME>/api/admin-j/lib/ct_admin_api.jar`
- and the SSL-related libraries, all located in `<CT_HOME>/lib` (required only if you are connecting over SSL):
`jcsi_base.jar`
`jcsi_provider.jar`
`jce1_2-do.jar`

```

jcert.jar
jnet.jar
jsse.jar
certj.jar
jsafe.jar
jsafeJCE.jar
rsajsse.jar
sslj.jar

```

Building for UNIX

To build RSA ClearTrust Administrative C API applications on Solaris, you will need **gcc** version 2.95.1 or higher. (To download gcc, see <http://gcc.gnu.org/>) Building requires the following resources:

- Link your code with the following libraries:
`lct_admin_api -lsocket -lnsl`
- Include the RSA ClearTrust API *headers* directory. (See “[Header Files](#)” on page [12](#) for the directory path.)
- Note also that a number of Java libraries will be required at runtime. See “[Initialization](#)” on page [15](#).

Building for Windows 2000 and NT

To build the RSA ClearTrust Administrative C API for Windows 2000 and NT, use Microsoft **Visual Studio** 6.0, or use GNU **gcc** version 2.95 or higher. Building requires the following resources:

- Link your code with the RSA ClearTrust Administrative API library. To use the DLL version of the library, include `ct_admin_api.dll`. To use the statically linked version, you would include `ct_admin_api.lib` in your link options.
- Include the RSA ClearTrust API *headers* directory. (See “[Header Files](#)” on page [12](#) for the directory path.)
- Note also that a number of Java libraries will be required at runtime. See “[Initialization](#)” on page [15](#).

Initialization and Login Operations

Initialization

Your client application will establish an RSA ClearTrust session by calling the `ct_initialize_api()` function of `ct_commands.h`. The initialization function links the application to the required Java libraries and connects to the Entitlements Server using the specified administrative user name and password.

`ct_initialize_api()` takes as an argument a reference to a `ct_map` containing a `CT_CLASSPATH_KEY` field set to a semicolon-separated list of all the required Java libraries. The complete path name is required for each library. The required libraries are:

- The Java Admin API JAR file, from the *Java* API directory:
`<CT_HOME>/api/admin-j/lib/ct_admin_api.jar`
- If you are connecting over SSL, the SSL-related JARs are required:
`<CT_HOME>/lib/jcsi_base.jar`
`<CT_HOME>/lib/jcsi_provider.jar`
`<CT_HOME>/lib/jce1_2-do.jar`
`<CT_HOME>/lib/jcert.jar`
`<CT_HOME>/lib/jnet.jar`
`<CT_HOME>/lib/jsse.jar`
`<CT_HOME>/lib/certj.jar`
`<CT_HOME>/lib/jsafe.jar`
`<CT_HOME>/lib/jsafeJCE.jar`
`<CT_HOME>/lib/rsajsse.jar`
`<CT_HOME>/lib/sslj.jar`

Typically, you will create a string similar to this example:

```
char *class_path_key = "c:\\ct\\api\\admin-j\\lib\\ct_admin_api.jar;
c:\\ct\\lib\\jcsi_base.jar; \
c:\\ct\\lib\\jcsi_provider.jar; \
c:\\ct\\lib\\jce1_2-do.jar; \
c:\\ct\\lib\\jcert.jar; \
c:\\ct\\lib\\jnet.jar; \
c:\\ct\\lib\\jsse.jar; \
c:\\ct\\lib\\certj.jar; \
c:\\ct\\lib\\jsafe.jar; \
c:\\ct\\lib\\jsafeJCE.jar; \
c:\\ct\\lib\\rsajsse.jar; \
c:\\ct\\lib\\sslj.jar";
```

Once you have created the `CT_CLASSPATH_KEY` and inserted it into the `ct_map`, you can call `ct_initialize_api()`, passing in the map, here called, "myMap." The function returns an `RC_OK` value if the initialization succeeds.

```
if ((rc = ct_initialize_api( myMap )) != RC_OK) {
    printf("The ct_initialize_api() call failed: %d\n", rc);
}
```

To see how `ct_initialize_api()` is used, see the complete example “[AdminUser.c](#)” on page 60.

Login

After you have initialized your application session, use the `ct_connect()` function to log into the Entitlements Manager. This function takes the following arguments, some of which may be left null:

- `srvr_name`: API Server's (Entitlements Server's) hostname or ip address
- `port`: API Server's port number
- `timeout`: seconds this connection will wait for a response before timing out. A typical setting would be 300 seconds.
- `use_ssl`: Whether to use SSL to connect to the API Server. See the following section.
- `admin_user`: Name of the administrative user connecting (optional)
- `admin_pw`: Administrative user's password (optional)
- `admin_role`: Name of the administrative user's administrative role (optional)
- `admin_group`: Name of the administrative user's administrative group (optional)

For an example containing `ct_connect()`, see “[AdminUser.c](#)” on page 60.

Connecting With and Without SSL

An Administrative API client may connect to the Entitlements Server as an authenticated SSL client, an anonymous SSL client, or as a non-SSL client. The `cleartrust.eserver.api_port.use_ssl` setting in the Entitlements Server's `eserver.conf` file indicates which type of connection is required for Administrative API clients. When writing your Administrative API programs, make sure that the boolean you pass as the `use_ssl` parameter of the `ct_connect` function matches the setting of `cleartrust.eserver.api_port.use_ssl`. Your settings will match one of the scenarios shown in the subsections that follow.

For more information, See the section “SSL Settings for RSA ClearTrust API Clients” in Chapter 7 of the *RSA ClearTrust Installation and Configuration Guide*.

On a System Running Clear Text Connections

If the RSA ClearTrust system is running with clear text connections between servers, as specified with:

```
cleartrust.net.ssl.use=false  
cleartrust.net.ssl.require_authentication=false
```

Then the Administrative API clients must also connect in clear text, as specified with:

```
cleartrust.eserver.api_port.use_ssl=false
```

On a System Running Anonymous SSL Connections

If the RSA ClearTrust system is running with anonymous SSL connections between servers, as specified with:

```
cleartrust.net.ssl.use=true  
cleartrust.net.ssl.require_authentication=false
```

Then the Administrative API clients can connect either via clear text:

```
cleartrust.eserver.api_port.use_ssl=false
```

or via anonymous SSL:

```
cleartrust.eserver.api_port.use_ssl=true
```

On a System Running Mutually Authenticated SSL Connections

If the RSA ClearTrust system is running with mutually authenticated SSL connections between servers, as specified with:

```
cleartrust.net.ssl.use=true  
cleartrust.net.ssl.require_authentication=true
```

Then the Administrative API clients can connect either via clear text:

```
cleartrust.eserver.api_port.use_ssl=false
```

or via mutually authenticated SSL:

```
cleartrust.eserver.api_port.use_ssl=true
```

The Functions of ct_commands.h

`ct_commands.h` is the main RSA ClearTrust Administrative API header file and contains all the API session initialization functions as well as commands for finding, loading and saving users, resources, and entitlements.

This section contains brief overviews of each function in `ct_commands.h`. For more detailed descriptions, see the comments in `<CT_HOME>/api/admin-c/include/ct_commands.h`.

Functions For Loading Objects

The following methods illustrate the typical object retrieving methods available in RSA ClearTrust. For most objects, you can look for the object based on name. In addition, `_by_index` and `_by_range` methods are provided that allow you to iterate over sets of objects. Please note the warning below regarding index numbers.

- `ct_get_num_of_users`
- `ct_get_user_and_properties`
- `ct_get_user_by_index`
- `ct_get_user_by_name`
- `ct_get_userprop_for_user_by_name`
- `ct_get_users_by_names`
- `ct_get_users_by_range`



Warning: The order in which objects are returned in a search is not defined. As a result, you cannot use the `get_*_by_index()` functions to reliably locate a particular item based on where you expected it to be placed in a set. Note that the `get_*_by_index()` functions *are useful* for iterating over all elements in the set. Similarly, you cannot use the `get_*_by_range()` functions to reliably locate a particular set of items, but you can use them to retrieve a set of data in segments or to retrieve the entire set.

Administrative Functions

Administrative functions create, edit, and retrieve RSA ClearTrust entitlements data.

Table 3.1 Administrative Functions

Function	Description
<code>ct_initialize_api</code>	Initializes the RSA ClearTrust API.
<code>ct_connect</code>	Connects to the RSA ClearTrust API Server.
<code>ct_connect_admin</code>	Deprecated. Use <code>ct_connect()</code> instead.

Table 3.1 Administrative Functions

Function	Description
ct_login	Login the administrative user to the RSA ClearTrust API Server.
ct_disconnect	Disconnects from the RSA ClearTrust API Server.
ct_free_adminroleid_array	Frees the CT_AdminRoleId array.
ct_get_adminroleids_for_user	Gets an array of AdminRoleIds, which represents all administrative roles that the user currently belongs to.

Password Setting Functions

The password setting functions allow an administrative user to reset users' passwords.

Table 3.2 Password Setting Functions

Function	Description
ct_force_password_expiration	Forces the expiration of the specified user password.
ct_get_password_expiration	Gets the expiration date of the specified user password.
ct_reset_password	Resets a user password.
ct_revert_password	Reverts the password expiration date to the value specified by the password policy associated with the user's administrative group.
ct_set_password	Sets a user password.
ct_set_password_expiration	Sets a user password expiration date.

Deprecated Runtime-Type Functions

These runtime-type functions have been deprecated in 4.7. While they will still work in this release, they will be removed in the near future. You should use the equivalent functions in ClearTrust Runtime API instead. See [Chapter 5, "Runtime C API"](#).

Table 3.3 Deprecated Runtime-Type Functions

Deprecated Function	Description	Replacement in Runtime API
ct_check_access	Deprecated. Checks user accessibility to a URI.	ct_authorize() or ct_authorize_pool()
ct_check_function	Deprecated. Checks user accessibility to an application's function.	ct_authorize() or ct_authorize_pool()
ct_check_password	Deprecated. Checks a user's password.	ct_authenticate() and ct_authenticate_pool()
ct_validate_user	Deprecated.	
ct_flush_cache	Deprecated. Flushes the RSA ClearTrust Authorizer caches.	ct_clear_server_caches() and ct_clear_server_caches_pool()

Administrative Objects

Administrative objects consists of the following:

- Administrative group (CT_AdminGroup) — Defines which administrative users own (can view and modify) a set of objects.
- Administrative user (CT_AdminUser) — A user dedicated only to RSA ClearTrust administration activities. An administrative user is not to be confused with an IUser who is granted or denied access to resources that are protected by RSA ClearTrust. IUsers cannot act as administrative users.
- Administrative role (CT_Admin) — A set of permissions defining what an administrative user logged in under this role can and cannot do.
- Password policy (CT_PasswordPolicy) — A set of restrictions on passwords for users.

The following sections describe the types of Administrative objects and the `ct_commands.h` functions related to each object type.

Administrative Group (VBU)

A CT_AdminGroup object represents an *administrative group*, which is a grouping of Administrative Roles. An Administrative Group is also a high level grouping of owned RSA ClearTrust entities. It is used to control the administrative users' abilities to view and modify these entities — specifically users, groups, applications, Web servers, server trees, and user property definitions. The ability to view and modify these entities can be restricted to those Administrative Roles contained within the owning Administrative Group. Finally, the Administrative Group helps determine the behavior of some of the entities it owns.

Table 3.4 CT_AdminGroup Object

Element	Type	Description
id	int	Reference for the API layer.
name	char *	Name of the administrative group.
description	char *	Textual description of the administrative group.
passwordpolicy	char *	Name of the password policy associated with this administrative group.
defaultPrivate	CT_BOOLEAN	Indicates whether objects created in this administrative group through the GUI are private or public by default.
forceExpiry	CT_BOOLEAN	indicates whether users created in this administrative group have expired passwords on creation. On first login, the user will be required to change the password.

The following table describes the `ct_commands.h` functions that operate on an administrative group object.

Table 3.5 Administrative Group Functions of `ct_commands.h`

Function	Description
<code>ct_create_admin_group</code>	Creates an administrative group object.
<code>ct_delete_admin_group</code>	Deletes an administrative group object.
<code>ct_get_admin_group_by_index</code>	Gets the administrative group for the an index number. Like all <code>get_by_index</code> methods, this method is generally not useful for retrieving a specific item. See page 18.
<code>ct_get_admingroup_by_name</code>	Gets the administrative group for the specified name.
<code>ct_get_admingroups_by_index</code>	Gets a range of administrative groups, based on an array of administrative group indexes.
<code>ct_get_admingroups_by_name</code>	Gets a range of administrative groups, based on an array of administrative group names.
<code>ct_get_admingroups_by_range</code>	Gets a range of administrative groups for the specified index range. Like all <code>get_by_range</code> methods, this method is generally not useful for retrieving a specific set of items. See page 18.
<code>ct_get_adminrole_for_admingroup_by_index</code>	Gets the administrative role for the specified index into the specified administrative group.
<code>ct_get_adminrole_for_admingroup_by_name</code>	Gets the administrative role for the specified administrative group and name.
<code>ct_get_adminroles_for_admingroup_by_names</code>	Gets an array of administrative roles for the specified administrative group and array of names.
<code>ct_get_adminroles_for_admingroup_by_range</code>	Gets the requested range of administrative roles contained in the administrative group for the index range specified.
<code>ct_get_application_for_admingroup_by_index</code>	Gets the requested application owned by the administrative group for the specified index.
<code>ct_get_application_for_admingroup_by_name</code>	Gets the application with the specified name for the administrative group specified.
<code>ct_get_applications_for_admingroup_by_names</code>	Gets an array of applications for the administrative group with the names specified.
<code>ct_get_applications_for_admingroup_by_range</code>	Gets an array of applications for the administrative group for the index range specified.
<code>get_apps_for_user</code>	The <code>get_apps_for_user()</code> method in the C API has been removed. In previous releases, one could get a list of a user's set of available resources by calling this method. This was possible because, in those releases of the product, access to every resource was provided by means of an application function associated with that resource. In version 4.7, access to a given resource may be provided by means of a an entitlement to that resource directly, or by means of an entitlement to an application that contains the resource. Since there are multiple ways of providing access to a given resource, the results returned by <code>get_apps_for_user()</code> would no longer provide a complete list of accessible resources. For this reason, the method has been removed.
<code>ct_get_group_for_admingroup_by_index</code>	Gets the group owned by the administrative group for the specified index.

Table 3.5 Administrative Group Functions of `ct_commands.h`

Function	Description
<code>ct_get_group_for_admingroup_by_name</code>	Gets the group owned by the administrative group for the specified group name.
<code>ct_get_groups_for_admingroup_by_names</code>	Gets an array of groups owned by the administrative group for the names specified.
<code>ct_get_groups_for_admingroup_by_range</code>	Gets an array of groups owned by the administrative group for the index range specified.
<code>ct_get_num_of_admingroups</code>	Gets the number of administrative groups in the entitlements database.
<code>ct_get_num_of_adminroles_for_admingroup</code>	Gets the number of administrative roles owned by the specified administrative group.
<code>ct_get_num_of_applications_for_admingroup</code>	Gets the number of applications owned by the administrative group specified.
<code>ct_get_num_of_groups_for_admingroup</code>	Gets the number of groups owned by the specified administrative group.
<code>ct_get_num_of_realms_for_admingroup</code>	Deprecated: Gets the number of realms owned by the specified administrative group.
<code>ct_get_num_of_userpropdefs_for_admingroup</code>	Gets the number of UserPropertyDefinitions owned by the specified administrative group.
<code>ct_get_num_of_users_for_admingroup</code>	Gets the number of users owned by the specified administrative group.
<code>ct_get_num_of_webservers_for_admingroup</code>	Gets the number of Web servers owned by the specified administrative group.
<code>ct_get_realm_for_admingroup_by_index</code>	Deprecated: Gets the realm owned by the administrative group for the index specified.
<code>ct_get_realm_for_admingroup_by_name</code>	Deprecated: Gets the realm owned by the administrative group for the name specified.
<code>ct_get_realms_for_admingroup_by_names</code>	Deprecated: Gets an array of realms owned by the administrative group for the names specified.
<code>ct_get_realms_for_admingroup_by_range</code>	Deprecated: Gets an array of realms owned by the administrative group for the index range specified.
<code>ct_get_user_for_admingroup_by_index</code>	Gets the user owned by the administrative group for the index specified.
<code>ct_get_user_for_admingroup_by_name</code>	Gets the user owned by the administrative group for the name specified.
<code>ct_get_userpropdef_for_admingroup_by_index</code>	Gets the UserPropertyDefinition owned by the administrative group for the index specified.
<code>ct_get_userpropdef_for_admingroup_by_name</code>	Gets the UserPropertyDefinition owned by the administrative group for the name specified.
<code>ct_get_userpropdefs_for_admingroup_by_names</code>	Gets an array of UserPropertyDefinitions owned by the administrative group for the names specified.

Table 3.5 Administrative Group Functions of `ct_commands.h`

Function	Description
<code>ct_get_userpropdefs_for_admingroup_by_range</code>	Gets an array of UserPropertyDefinitions owned by the administrative group for the index range specified.
<code>ct_get_users_for_admingroup_by_names</code>	Gets an array of users owned by the administrative group for the names specified.
<code>ct_get_users_for_admingroup_by_range</code>	Gets an array of users owned by the administrative group for the index range specified.
<code>ct_get_webserver_for_admingroup_by_index</code>	Gets the Web server owned by the administrative group for the index specified.
<code>ct_get_webserver_for_admingroup_by_name</code>	Gets the Web server owned by the administrative group for the name specified.
<code>ct_get_webservers_for_admingroup_by_name</code>	Gets an array of Web servers owned by the administrative group for the names specified.
<code>ct_get_webservers_for_admingroup_by_range</code>	Gets an array of Web servers owned by the administrative group for the index range specified.
<code>ct_save_admin_group</code>	Saves the administrative group to the database.
<code>ct_transfer_ownership</code>	Transfers the ownership from one administrative group to another administrative group

Administrative User

Administrative users (CT_AdminUsers) are users that are used purely for RSA ClearTrust administration activities, and are not users that are granted access to resources that are protected by RSA ClearTrust. In practice, administrative users typically exist entirely within the RSA ClearTrust policy repository (as opposed to a separate user store in LDAP or elsewhere), but this is configurable. This functionality existed on CT_User in previous releases, but has since been separated to more easily support read-only user stores.

Administrative users cannot be granted entitlements to RSA ClearTrust-protected resources. If you have an administrative user who wishes to access resources, you must create a separate CT_User account for that person. See “Users” on page 31.

An administrative user can view (and usually edit) public records and records owned by his administrative group (see page 20), and his actions are limited to those permitted by his administrative role (see page 25).

Table 3.6 Administrative User (CT_AdminUser) Object

Element	Type	Description
id	object	Reference for the API layer. Do not modify!
admin_id	object	Reference for the API layer. Do not modify!
admin_name	char*	Name of the owning administrative group.
ct_public	CT_BOOLEAN	If true, object is modifiable by any Administrator
name	char*	This is the name of the entity
firstname	char*	First name of admin user
lastname	char*	Last name of admin user
emailaddr	char*	E-mail address of admin user
superuser	CT_BOOLEAN	If true and user belongs to an Administrative Role, user has full access to the ClearTrust database.
superHelpDesk	CT_BOOLEAN	If true, user can modify passwords for all ClearTrust users
password	char*	Password of the user, used only to set the password, it is not set when a admin user is retrieved from the server.
is_locked	CT_BOOLEAN	If true, the admin user is locked out from the system. This means the admin user cannot administer ClearTrust.
startdate	DATE_TYPE	Start date of the user
enddate	DATE_TYPE	End date of the user
password_expiration_date	DATE_TYPE	The date at which the current password expires. Beyond this date the admin user will not be able to log into the system until the password has been changed.

Administrative Role

An CT_Admin object represents an *administrative role*. This role is a set of permissions defining what an administrative user logged in under this role can and cannot do.

Table 3.7 CT_Admin (Administrative Role) Object

Element	Type	Description
id	int	Reference for the API layer.
name	char *	Name of the administrative role used for object reference.
description	char *	Textual description of the administrative role.
admin_id	int	Reference for the API layer.
add_user	CT_BOOLEAN	Permission to add a user. True means this role allows the action; false means it forbids the action.
mod_user	CT_BOOLEAN	Permission to modify a user.
del_user	CT_BOOLEAN	Permission to delete a user.
add_admin_user	CT_BOOLEAN	Permission to add an administrative user.
mod_admin_user	CT_BOOLEAN	Permission to modify an administrative user.
del_admin_user	CT_BOOLEAN	Permission to delete an administrative user.
add_group	CT_BOOLEAN	Permission to add a group.
mod_group	CT_BOOLEAN	Permission to modify a group.
del_group	CT_BOOLEAN	Permission to delete a group.
add_realm	CT_BOOLEAN	Deprecated: Permission to add a realm.
mod_realm	CT_BOOLEAN	Deprecated: Permission to modify a realm.
del_realm	CT_BOOLEAN	Deprecated: Permission to delete a realm.
add_app	CT_BOOLEAN	Permission to add an application.
mod_app	CT_BOOLEAN	Permission to modify an application.
del_app	CT_BOOLEAN	Permission to delete an application.
add_server	CT_BOOLEAN	Permission to add a server.
mod_server	CT_BOOLEAN	Permission to modify a server.
del_server	CT_BOOLEAN	Permission to delete a server.
add_user_prop_def	CT_BOOLEAN	Permission to add a user property definition.
mod_user_prop_def	CT_BOOLEAN	Permission to modify a user property definition.
del_user_prop_def	CT_BOOLEAN	Permission to delete a user property definition.
add_admin	CT_BOOLEAN	Permission to add an administrative role
mod_admin	CT_BOOLEAN	Permission to modify an administrative role.

Table 3.7 CT_Admin (Administrative Role) Object

Element	Type	Description
del_admin	CT_BOOLEAN	Permission to delete an administrative role.
set_password	CT_BOOLEAN	Permission to reset a user's password.

The following table describes the `ct_commands.h` functions that operate on the administrative role object.

Table 3.8 CT_Admin (Administrative Role) Functions of `ct_commands.h`

Function	Description
<code>ct_add_admin_user_to_admin_role</code>	Adds an administrative user to the specified administrative role.
<code>ct_create_admin_role</code>	Creates an administrative role associated with an administrative group.
<code>ct_delete_admin_role</code>	Deletes the specified administrative role.
<code>ct_get_admin_user_in_admin_role_by_index</code>	Given an administrative role, and an index, returns requested user who is a member of the administrative role.
<code>ct_get_num_admin_users_in_admin_role</code>	Gets the number of users that are members of the administrative role.
<code>ct_get_adminrole_for_admingroup_by_name</code>	Given an administrative group, and name, returns the administrative role with the specified name and the administrative role is contained in the administrative group.
<code>ct_get_adminroles_for_admingroup_by_names</code>	Given an administrative group, and an array of names, returns an array of administrative roles with the specified names, and the administrative roles is contained in the administrative group.
<code>ct_get_adminroles_for_admingroup_by_range</code>	Given an administrative group, start index, and end index, returns the requested range of administrative roles contained in the administrative group.
<code>ct_get_admin_user_in_admin_role_by_index</code>	Gets the user who is a member of the administrative role for the specified index.
<code>ct_get_admin_user_in_admin_role_by_name</code>	Gets the user who is a member of the administrative role for the specified name.
<code>ct_get_admin_users_in_admin_role_by_names</code>	Gets an array of users who are members of the administrative role for the names specified.
<code>ct_get_admin_users_in_admin_role_by_range</code>	Gets an array of users who are members of the administrative role for the index range specified.
<code>ct_get_context_admin_role</code>	Retrieves the administrative role from the current session or from the given context.
<code>ct_remove_admin_user_from_admin_role</code>	Removes an administrative user from the administrative role.
<code>ct_save_admin_role</code>	Saves the administrative role object in the database.

Password Policy

A *password policy* object (CT_PasswordPolicy) is a set of restrictions on passwords for users. Each administrative group has an associated password policy that is applied to users owned by that administrative group.

Table 3.9 CT_PasswordPolicy Object

Element	Type	Description
id	int	Reference for the API layer.
name	char *	Name of the password policy.
description	char *	Textual description of the password policy.
min_length	int	Password minimum length as defined by this policy.
max_length	int	Password maximum length as defined by this policy.
dictionary_file	char *	Dictionary file name for this policy.
non_letter	CT_BOOLEAN	Indicates whether the password policy requires that passwords contain a non-letter character.
excluded_chars	char *	Characters excluded from passwords.
lifetime	char *	Default lifetime for user passwords.
history	int	Number of past passwords to exclude.

The following table describes the `ct_commands.h` functions that handle password policies.

Table 3.10 Password Policy Functions of `ct_commands.h`

Function	Description
<code>ct_create_password_policy</code>	Creates a PasswordPolicy object.
<code>ct_delete_password_policy</code>	Deletes a PasswordPolicy object.
<code>ct_get_default_password_policy</code>	Gets the default PasswordPolicy.
<code>ct_get_num_of_password_policies</code>	Gets the number of PasswordPolicies in the entitlement server database.
<code>ct_get_password_policies_by_names</code>	Gets an array of PasswordPolicies for the names specified.
<code>ct_get_password_policies_by_range</code>	Gets an array of PasswordPolicies for the index range specified.
<code>ct_get_password_policy_by_index</code>	Gets the PasswordPolicy for the index specified.
<code>ct_get_password_policy_by_name</code>	Gets the PasswordPolicy for the name specified.
<code>ct_save_password_policy</code>	Saves the PasswordPolicy in the database.
<code>set_default_password_policy</code>	Sets the default PasswordPolicy in the database.
<code>get_default_password_policy</code>	Gets the default password policy.
<code>transfer_admingroup</code>	Transfers ownership of the administrative group to another administrative group.

Participants

Participant objects model the people and organizations whose access to resources is governed by the RSA ClearTrust system.

- Group: a collection of users
- User: an end-user person who will, upon successful authentication and authorization, be given access to RSA ClearTrust-protected resources.
- User property: an extra detail about a user that can be used as a criterion for access decisions, for Web personalization, etc. A user property is stored in a field that has been declared and defined in a user property definition.
- User property definition: Mechanism for adding extra data fields to user records. In order to add a data field that is usable on all user records, you must create a user property definition that establishes the name and datatype. Once you have created and saved the user property definition, you can begin storing data in the new user property field.



When creating users and groups, please note that their names may not contain any of the following characters: “,”, “+”, “””, “\”, “<”, “>” or “;”.

CT_EntityHdr Struct

The CT_EntityHdr struct contains common attributes for groups and users.

Table 3.11 CT_EntityHdr

Element	Type	Description
id	int	Reference for the API layer.
admin_id	int	Reference for the API layer.
admin_name	char *	Name of the owning administrative group.
creationdate	DATE_TYPE	Deprecated. The creation date of the object.
ct_public	CT_BOOLEAN	Determines whether the object is visible to and editable by all administrative users.
name	char *	The name of the entity.

Groups

A *group* object (CT_Group) represents a collection of users and/or other groups. Any user or group can be included in many groups.



Note: If your installation uses collections of groups, please note that the mechanism for doing this has changed with the release of RSA ClearTrust 4.7. Previously, RSA ClearTrust provided the realm object (CT_Realm struct) for building collections of groups. In version 4.7, the IRealm interface is deprecated, and this functionality has been replaced with nested groups. This means that a group may contain other groups, which may in turn contain other groups, and so on. The deprecation of IRealm means that, while groups may still be collected into realms in 4.7, the IRealm interface will no longer exist in the next version of RSA ClearTrust.

Table 3.12 CT_Group Object

Element	Type	Description
hdr	CT_EntityHdr	Header of the object.
description	char *	Textual description of the group.

The following table describes the `ct_commands.h` functions that can operate on a group object.

Table 3.13 Group Functions of `ct_commands.h`

Function	Description
<code>ct_add_user_to_group</code>	Adds a user to a group.
<code>ct_create_entitlement_for_group</code>	Creates a basic entitlement between a group and an application function.
<code>ct_create_group</code>	Creates a group object. Group names may not contain any of the following characters: “,” “+”, “””, “\”, “<”, “>” or “.”.
<code>ct_delete_group</code>	Deletes a group object.
<code>ct_get_entitlement_for_group</code>	Gets a basic entitlement between a group and an application function.
<code>ct_get_exp_entitlement_for_group_by_index</code>	Gets the basic entitlement assigned to the group for the specified index.
<code>ct_get_exp_entitlements_for_group_by_range</code>	Gets an array of basic entitlements assigned to the group for the specified index range.
<code>ct_get_group_by_index</code>	Gets the group for the specified index.
<code>ct_get_group_by_name</code>	Gets the group for the specified name.
<code>ct_get_group_owner</code>	Gets the owner of a group.
<code>ct_get_groups_by_names</code>	Gets an array of groups for the array of names specified.
<code>ct_get_groups_by_range</code>	Gets an array of groups for the index range specified.

Table 3.13 Group Functions of `ct_commands.h`

Function	Description
<code>ct_get_num_of_exp_entitlements_for_group</code>	Gets the number of basic entitlements assigned to the group.
<code>ct_get_num_of_groups</code>	Gets the number of groups in the entitlement server database.
<code>ct_get_num_of_realms_for_group</code>	Deprecated: Gets the number of realms that are parents of the group.
<code>ct_get_num_of_users_for_group</code>	Gets the number of users that are children of the group.
<code>ct_get_realm_for_group_by_index</code>	Deprecated: Gets the realm of the group for the specified index.
<code>ct_get_realm_for_group_by_name</code>	Deprecated: Gets the realm of the group for the specified name.
<code>ct_get_realms_for_group_by_names</code>	Deprecated: Gets an array of realms of the group for the specified names.
<code>ct_get_realms_for_group_by_range</code>	Deprecated: Gets an array of realms of the group for the specified index range.
<code>ct_get_user_for_group_by_index</code>	Gets the user that is a child of the group for the specified index.
<code>ct_get_user_for_group_by_name</code>	Gets the user that is a child of the group for the specified name.
<code>ct_get_users_for_group_by_names</code>	Gets an array of users that are children of the group for the specified names.
<code>ct_get_users_for_group_by_range</code>	Gets an array of users that are children of the group for the specified index range.
<code>ct_remove_user_from_group</code>	Removes a user from a group.
<code>ct_save_group</code>	Saves the group to the database.
<code>ct_set_group_owner</code>	Sets the administrative group that owns the group.

Users

A *user* object (CT_User) represents a user who will attempt to view or use an RSA ClearTrust-protected URL or other resource. Users are usually collected into Groups and are given rights to resources via basic entitlements and SmartRules. A user may be a member of many groups.



Warning: Do not confuse users with administrative users; they are separate and unrelated objects. See [“Administrative User”](#) on page 24.

The following table describes the elements of the user object.

Table 3.14 CT_User Object

Element	Type	Description
hdr	CT_EntityHdr	Header of the object.
firstname	char *	First name of the user.
lastname	char *	Last name of the user.
emailaddr	char *	email address of the user.
superuser	CT_BOOLEAN	Deprecated. Applies only to the newly-introduced CT_AdminUser.
superHelpDesk	CT_BOOLEAN	Deprecated. Applies only to the newly-introduced CT_AdminUser.
password	char *	Password of the user, used only to set the password. It is not set when a user is retrieved from the server.
dn	char *	Distinguished Name of the user. This attribute is used to map an RSA ClearTrust user to a user in an external directory. user names may not contain any of the following characters: “ ”, “+”, “@”, “\”, “<”, “>” or “;”.
is_locked	CT_BOOLEAN	If true, the user is locked out from the system. This means that the user is denied access to any resource.
startdate	DATE_TYPE	Start date of the user.
enddate	DATE_TYPE	End date of the user.
propArray	ObjectArrayRef	Array of user properties.

Relationships are not defined within the user object. You access the other objects through the relationship APIs.

The following table describes the `ct_commands.h` functions that can operate on a user object.

Table 3.15 User Functions of `ct_commands.h`

Function	Description
<code>ct_create_entitlement_for_user</code>	Creates a basic entitlement between a user and an application function.
<code>ct_create_user_and_properties</code>	Creates a user and all of its properties. Note that user names may not contain any of the following characters: “,”, “+”, “””, “\”, “<”, “>” or “.”.
<code>ct_delete_user</code>	Deletes a user.
<code>ct_get_entitlement_for_user</code>	Gets a basic entitlement between a user and an application function.
<code>ct_get_exp_entitlement_for_user_by_index</code>	Gets the basic entitlement assigned to the user for the index specified.
<code>ct_get_exp_entitlements_for_user_by_range</code>	Gets an array of basic entitlements assigned to the user for the specified index range.
<code>ct_get_group_for_user_by_index</code>	Gets the parent group for this user for the specified index.
<code>ct_get_group_for_user_by_name</code>	Gets the parent group for this user for the specified name.
<code>ct_get_groups_for_user_by_names</code>	Gets an array of parent groups for this user for the specified names.
<code>ct_get_groups_for_user_by_range</code>	Gets an array of parent groups for this user for the specified index range.
<code>ct_get_num_of_exp_entitlements_for_user</code>	Gets the number of basic entitlements assigned to the user.
<code>ct_get_num_of_groups_for_user</code>	Gets the number of parent groups for which this user is a member.
<code>ct_get_num_of_userprops_for_user</code>	Gets the number of UserProperties associated with the user.
<code>ct_get_num_of_users</code>	Gets the number of users in the entitlement server database.
<code>ct_get_user_and_properties</code>	Gets the specified user and all of its properties.
<code>ct_get_user_and_properties_by_dn</code>	Gets the user specified by its Distinguished Name and all of its properties.
<code>ct_get_user_by_index</code>	Gets the user for the specified index.
<code>ct_get_user_by_name</code>	Gets the user for the specified name.
<code>ct_get_user_owner</code>	Gets the owner of the specified user.
<code>ct_get_userprop_for_user_by_index</code>	Gets the UserProperty associated with the user for the specified index.
<code>ct_get_userprop_for_user_by_name</code>	Gets the UserProperty associated with the user for the specified name.
<code>ct_get_userprops_for_user_by_names</code>	Gets an array of UserProperties associated with the user for the specified names.
<code>ct_get_userprops_for_user_by_range</code>	Gets an array of UserProperties associated with the user for the specified index range.
<code>ct_get_users_by_names</code>	Gets an array of users for the specified names.
<code>ct_get_users_by_range</code>	Gets an array of users for the specified index range.
<code>ct_save_user</code>	Saves the specified user in the database.

Table 3.15 User Functions of `ct_commands.h`

Function	Description
<code>ct_save_user_and_properties</code>	Saves a user and all of its properties in the database.
<code>ct_set_user_owner</code>	Sets the owner (administrative group) of the user.

User Properties

A *user property* object (`CT_UserProperty`) is an extra detail about a user that can be used as a criterion for access decisions, for Web personalization, etc. For example, based on a user property, a SmartRule can calculate a user's accessibility to an application. A user property is stored in a field that has been declared and defined in a *user property definition*.

Table 3.16 `CT_UserProperty` Object

Element	Type	Description
<code>id</code>	<code>int</code>	Reference for the API layer.
<code>user_id</code>	<code>int</code>	User ID of the UserProperty.
<code>name</code>	<code>char *</code>	Deprecated. The name is available on <code>CT_UserPropertyDefinition</code> .
<code>type</code>	<code>CT_USERPROP_DATATYPE</code>	Defines the data type of the UserProperty.
<code>val</code>	<code>CT_USERPROP_DATAVALUE</code>	Value of the data type.
<code>isSet</code>	<code>CT_BOOLEAN</code>	Indicates whether this UserProperty is set in the RSA ClearTrust database.

User Property Definitions

A *user property definition* object (`CT_UserPropertyDefinition`) declares an extra data field that may be populated in any user record. In order to add a data field that is usable on all user records, you must create a user property definition that establishes the name and datatype. Once you have created and saved the user property definition, you can begin storing data in the new user property field. Each value you store is a *user property* object (see previous section).

Table 3.17 `CT_UserPropertyDefinition` Object

Element	Type	Description
<code>id</code>	<code>int</code>	Reference for the API layer.
<code>admin_id</code>	<code>int</code>	Reference for the API layer.
<code>admin_name</code>	<code>char *</code>	Name of the owning administrative group.
<code>ct_public</code>	<code>CT_BOOLEAN</code>	Indicates whether the UserPropertyDefinition is visible to all RSA ClearTrust administrators.

Table 3.17 CT_UserPropertyDefinition Object

Element	Type	Description
name	char *	Name of the UserPropertyDefinition.
type	CT_USERPROP_DATATYPE	Data type of the UserPropertyDefinition.
description	char *	Textual description of the property definition.
source	char *	For internal use only.
readonly	CT_BOOLEAN	Indicates whether the property can be modified by all RSA ClearTrust administrators.
helpDeskAccessible	CT_BOOLEAN	Indicates whether this property can be seen by the superhelpdesk administrative user.
exportable	CT_BOOLEAN	Indicates whether this property is visible to Runtime API clients.

The following table describes the `ct_commands.h` functions that work with user property definition objects.

Table 3.18 User Property Definition Functions of `ct_commands.h`

Function	Description
<code>ct_create_user_property_definition</code>	Creates a UserPropertyDefinition.
<code>ct_delete_userpropdef</code>	Deletes the specified UserPropertyDefinition.
<code>ct_get_num_of_userpropdefs</code>	Gets the number of UserPropertyDefinitions in the entitlement database.
<code>ct_get_userpropdef_by_index</code>	Gets the UserPropertyDefinition for the specified index.
<code>ct_get_userpropdef_by_name</code>	Gets the UserPropertyDefinition for the specified name.
<code>ct_get_userpropdef_owner</code>	Gets the owner (administrative group) of the specified UserPropertyDefinition.
<code>ct_get_userpropdefs_by_names</code>	Gets an array of UserPropertyDefinitions for the specified names.
<code>ct_get_userpropdefs_by_range</code>	Gets an array of UserPropertyDefinitions for the specified index range.
<code>ct_save_userpropdef</code>	Saves the UserPropertyDefinition in the database.
<code>ct_set_userpropdef_owner</code>	Sets the owner (administrative group) of the specified UserPropertyDefinition.

Deprecated Structure: Realms

The deprecated *realm* object (CT_Realm) was used in previous releases to represent a collection of groups. As of 4.7, groups may be collected together in other groups, so realms are no longer needed. See “Groups” on page 29.

Table 3.19 The deprecated CT_Realm Object

Element	Type	Description
hdr	CT_EntityHdr	Header of the deprecated realm object.
description	char *	Textual description of the group.

Table 3.20 Deprecated Realm Functions of `ct_commands.h`. These are replaced by group functions.

Function	Description
<code>ct_add_group_to_realm</code>	Deprecated: Adds a group to a realm
<code>ct_create_entitlement_for_realm</code>	Deprecated: Creates a basic entitlement between a realm and an application function.
<code>ct_create_realm</code>	Deprecated: Creates a realm object. Realm names may not contain any of the following characters: “,” “+”, “””, “\”, “<”, “>” or “.”.
<code>ct_delete_realm</code>	Deprecated: Deletes the specified realm object.
<code>ct_get_entitlement_for_realm</code>	Deprecated: Gets a basic entitlement between a realm and an application function.
<code>ct_get_exp_entitlement_for_realm_by_index</code>	Deprecated: Gets the basic entitlement assigned to the realm for the specified index.
<code>ct_get_exp_entitlements_for_realm_by_range</code>	Deprecated: Gets an array of basic entitlements assigned to the realm for the specified index range.
<code>ct_get_group_for_realm_by_index</code>	Deprecated: Gets the group that is a child of the realm for the specified index.
<code>ct_get_group_for_realm_by_name</code>	Deprecated: Gets the group that is a child of the realm for the specified name.
<code>ct_get_groups_for_realm_by_names</code>	Deprecated: Gets an array of groups that are children of the realm for the specified names.
<code>ct_get_groups_for_realm_by_range</code>	Deprecated: Gets an array of groups that are children of the realm for the specified index range.
<code>ct_get_num_of_exp_entitlements_for_realm</code>	Deprecated: Gets the number of basic entitlements assigned to the specified realm.
<code>ct_get_num_of_groups_for_realm</code>	Deprecated: Gets the number of groups that are children of the realm.
<code>ct_get_num_of_realms</code>	Deprecated: Gets the number of realms in the entitlements database.
<code>ct_get_realm_by_index</code>	Deprecated: Gets the realm for the specified index.
<code>ct_get_realm_by_name</code>	Deprecated: Gets the realm for the specified name.

Table 3.20 Deprecated Realm Functions of `ct_commands.h`. These are replaced by group functions.

Function	Description
<code>ct_get_realm_owner</code>	Deprecated: Gets the owner of the realm.
<code>ct_get_realms_by_names</code>	Deprecated: Gets an array of realms specified by names.
<code>ct_get_realms_by_range</code>	Deprecated: Gets an array of realms for the specified index range.
<code>ct_remove_group_from_realm</code>	Deprecated: Removes a group from a realm.
<code>ct_save_realm</code>	Deprecated: Saves the realm to the database.
<code>ct_set_realm_owner</code>	Deprecated: Sets the realm's owner.

Policy Objects

Policy objects describe entitlements and rules that allow participants access to resources.

- Basic entitlement — Governs access to application functions based on a user's name or his or her membership in a group.
- SmartRule — Governs access to application functions based on user properties.

Basic Entitlements

An *explicit entitlement* object (`CT_ExplicitEntitlement`) defines a user's or group's access to a protected resource. Explicit entitlements are called *basic entitlements*. In order to create a basic entitlement, you must specify its entity ID (the ID of the user or group) and its application ID (the ID of the protected resource, which is an application, application function, or URL). The entitlement may grant or deny the user or group permission to access the application.

Basic entitlements granted to a group apply to all users in that group. Basic entitlements have a hierarchy; a basic entitlement granted or denied at the user level overrides any conflicting entitlements at the group level. Likewise, a basic entitlement granted or denied at the group level overrides any conflicting entitlement set on a parent group of that group.

In other words, when using nested group structures, if a user has a basic entitlement assigned at the level of a group and also has a conflicting entitlement assigned at the level of a second group that is higher up the group/parent group hierarchy, then the entitlement of the lowest-level group will take precedence. The reason for this is that the lowest-level group is thought of as providing the most precise definition of the user's privileges.

For example, if group `BronzeUsers` is denied access to application `CarLoanCalculator`, and the `BronzeUsers` group in turn contains a number of groups, then all the users in all the sub-groups of `BronzeUsers` are denied access. If you wished to override this denial for some users, there are two ways you could do it. One approach would be to create a new group (say, group `BronzeSpecial`) for those users, make that group a

member of the BronzeUsers group, and give that group access to CarLoanCalculator. The BronzeSpecial setting will override the BronzeUsers setting. The other approach would be to grant access at the individual user level. For example, if you grant user TBradshaw access to application CarLoanCalculator, this overrides any denials for TBradshaw defined in the groups to which he belongs.

Table 3.21 CT_ExplicitEntitlement Object

Element	Type	Description
id	int	Reference for the API layer.
accessible	CT_BOOLEAN	Indicates whether the application is accessible.

The following `ct_commands.h` functions operate on an explicit (basic) entitlement object.

Table 3.22 Basic Entitlement Functions of `ct_commands.h`

Function	Description
<code>ct_create_entitlement_for_user</code>	Deprecated: Creates a basic entitlement between a user and an application function.
<code>ct_create_entitlement_for_group</code>	Deprecated: Creates a basic entitlement between a group and an application function.
<code>ct_create_entitlement_for_group_and_appfunc</code>	Creates an explicit entitlement between a group and an application function.
<code>ct_create_entitlement_for_group_and_application</code>	Creates an explicit entitlement between a group and an application.
<code>ct_create_entitlement_for_group_and_appurl</code>	Creates an explicit entitlement between a group and an application URL.
<code>ct_create_entitlement_for_realm</code>	Deprecated: Creates a basic entitlement between a realm and an application function.
<code>ct_create_entitlement_for_realm_and_appfunc</code>	Deprecated in 4.7.
<code>ct_create_entitlement_for_realm_and_application</code>	Deprecated in 4.7.
<code>ct_create_entitlement_for_realm_and_appurl</code>	Deprecated in 4.7.
<code>ct_create_entitlement_for_user_and_appfunc</code>	Creates an explicit entitlement between a user and an application function.
<code>ct_create_entitlement_for_user_and_application</code>	Creates an explicit entitlement between a user and an application.
<code>ct_create_entitlement_for_user_and_appurl</code>	Creates an explicit entitlement between a user and an application URL.
<code>ct_create_entitlement_for_user</code>	Deprecated in 4.7.
<code>ct_delete_exp_entitlement</code>	Deletes the specified basic entitlement.
<code>ct_get_application_for_entitlement</code>	Gets the Application related to the Explicit Entitlement.
<code>ct_get_appfunc_for_entitlement</code>	Gets the application function related to the basic entitlement.

Table 3.22 Basic Entitlement Functions of `ct_commands.h`

Function	Description
<code>ct_get_appurl_for_entitlement</code>	Gets the Application URL related to the Explicit Entitlement.
<code>ct_get_entitlement</code>	Gets a basic entitlement between a Participant (user or group) and an application function.
<code>ct_get_entitlement_for_group.</code>	Deprecated in 4.7
<code>ct_get_entitlement_for_user.</code>	Deprecated in 4.7
<code>ct_get_entitlement_for_group_and_appfunc</code>	Retrieves an explicit entitlement between a CT_Group and a Application Function.
<code>ct_get_entitlement_for_group_and_application</code>	Retrieves an explicit entitlement between a CT_Group and a Application.
<code>ct_get_entitlement_for_group_and_appurl</code>	Retrieves an explicit entitlement between a CT_Group and a Application URL.
<code>ct_get_exp_entitlement_for_group_by_index</code>	Given a Group, and an index, returns requested ExplicitEntitlement assigned to the Group.
<code>ct_get_exp_entitlements_for_group_by_range</code>	Given a Group, start index, and end index, returns the requested range of ExplicitEntitlements assigned to the Group.
<code>ct_get_entitlement_for_realm</code>	Deprecated in 4.7.
<code>ct_get_entitlement_for_realm_and_appfunc</code>	Deprecated in 4.7.
<code>ct_get_entitlement_for_realm_and_application</code>	Deprecated in 4.7.
<code>ct_get_entitlement_for_realm_and_appurl</code>	Deprecated in 4.7.
<code>ct_get_entitlement_for_user_and_appfunc</code>	Retrieves an explicit entitlement between a user and an application function.
<code>ct_get_entitlement_for_user_and_application</code>	Retrieves an explicit entitlement between a user and an application.
<code>ct_get_entitlement_for_user_and_appurl</code>	Retrieves an explicit entitlement between a user and an application URL.
<code>ct_get_exp_entitlement_for_appfunc_by_index</code>	Given an Application Function, and an index, returns requested ExplicitEntitlement assigned to the Application Function.
<code>ct_get_exp_entitlements_for_appfunc_by_range</code>	Given a Application Function, start index, and end index, returns the requested range of ExplicitEntitlements assigned to the Application Function.
<code>ct_get_exp_entitlement_for_application_by_index</code>	Given an Application, and an index, returns requested ExplicitEntitlement assigned to the Application.
<code>ct_get_exp_entitlements_for_application_by_range</code>	Given a Application, start index, and end index, returns the requested range of ExplicitEntitlements assigned to the Application.
<code>ct_get_exp_entitlement_for_appurl_by_index</code>	Given an Application URL, and an index, returns requested ExplicitEntitlement assigned to the Application URL.
<code>ct_get_exp_entitlements_for_appurl_by_range</code>	Given a ApplicationURL, start index, and end index, returns the requested range of ExplicitEntitlements assigned to the ApplicationURL.

Table 3.22 Basic Entitlement Functions of `ct_commands.h`

Function	Description
<code>ct_get_exp_entitlement_for_user_by_index</code>	Given a user, and an index, returns requested entitlement assigned to the user.
<code>ct_get_exp_entitlements_for_user_by_range</code>	Given a user, start index, and end index, returns the requested range of entitlements assigned to the user.
<code>ct_get_exp_entitlement_for_realm_by_index</code>	Deprecated.
<code>ct_get_exp_entitlements_for_realm_by_range</code>	Deprecated.
<code>ct_get_group_for_entitlement</code>	Gets the group related to the basic entitlement.
<code>ct_get_num_of_exp_entitlements_for_application</code>	Given an Application, returns the number of ExplicitEntitlements assigned to the Application's 'ACCESS' ApplicationFunction.
<code>ct_get_num_of_exp_entitlements_for_appfunc</code>	Given an Application Function, returns the number of ExplicitEntitlements assigned to the Application Function.
<code>ct_get_num_of_exp_entitlements_for_appurl</code>	Given an Application URL, returns the number of ExplicitEntitlements assigned to the Application URL.
<code>ct_get_num_of_exp_entitlements_for_group</code>	Given a Group, returns the number of ExplicitEntitlements assigned to the Group.
<code>ct_get_num_of_exp_entitlements_for_realm</code>	Deprecated.
<code>ct_get_num_of_exp_entitlements_for_user</code>	Given a User, returns the number of entitlements assigned to the user.
<code>ct_get_realm_for_entitlement</code>	Deprecated: Gets the realm related to the basic entitlement.
<code>ct_get_user_for_entitlement</code>	Gets the user related to the basic entitlement
<code>ct_save_exp_entitlement</code>	Saves the basic entitlement to the database.

SmartRules

A *SmartRule* object (CT_SmartRule) governs access to a resource based on a user's value for a specified UserPropertyDefinition. The following table describes the elements of a SmartRule object.

Table 3.23 CT_SmartRule Object

Element	Type	Description
id	int	SmartRule primary key.
propDef_id	int	User property definition primary key.
appFunc_id	int	application function primary key.
type	CT_CRITERIA_DATATYPE	The type of SmartRule.
critValue	CT_CRITERIA_DATAVALUE	Data structure that holds the operator and value.
category	char *	The SmartRule category. Must be one of the following: "ALLOW" "DENY" "REQUIRE"

The following table describes the `ct_commands.h` operations that can be performed on a SmartRule object.

Table 3.24 SmartRule Functions of `ct_commands.h`

Function	Description
<code>ct_create_smart_rule</code>	Creates a SmartRule object.
<code>ct_create_smart_rule_for_app_func</code>	Creates a SmartRule associated with a UserPropertyDefinition and an ApplicationFunction.
<code>ct_create_smart_rule_for_app_url</code>	Creates a SmartRule associated with a UserPropertyDefinition and an ApplicationFunction.
<code>ct_create_smart_rule_for_application</code>	Creates a SmartRule associated with a UserPropertyDefinition and an ApplicationFunction.
<code>ct_delete_smart_rule</code>	Deletes a SmartRule object.
<code>ct_get_appfunc_for_smartrule</code>	Get the related ApplicationFunction for a SmartRule object.
<code>ct_get_application_for_smartrule</code>	Get the related Application for a SmartRule object.
<code>ct_get_appurl_for_smartrule</code>	Get the related Application URL for a SmartRule object.
<code>ct_get_num_of_smartrules_for_appfunc</code>	Gets the number of SmartRules associated with the specified ApplicationFunction.
<code>ct_get_num_of_smartrules_for_appfunc</code>	Given an ApplicationFunction, returns the number of SmartRules associated with the ApplicationFunction.
<code>ct_get_num_of_smartrules_for_application</code>	Given an Application, returns the number of SmartRules associated with the Application.
<code>ct_get_num_of_smartrules_for_appurl</code>	Given an ApplicationURL, returns the number of SmartRules associated with the ApplicationURL.

Table 3.24 SmartRule Functions of `ct_commands.h`

Function	Description
<code>ct_get_smartrules_for_appfunc_by_index</code>	Gets the SmartRule for the ApplicationFunction for the specified index.
<code>ct_get_smartrules_for_appfunc_by_index</code>	Given a ApplicationFunction, and an index, returns requested SmartRule associated with the ApplicationFunction.
<code>ct_get_smartrules_for_appfunc_by_range</code>	Gets an array of SmartRules for the ApplicationFunction for the specified index range.
<code>ct_get_smartrules_for_appfunc_by_range</code>	Given an ApplicationFunction, start index, and end index, returns the requested range of SmartRule contained in the ApplicationFunction.
<code>ct_get_smartrules_for_application_by_index</code>	Given a Application, and an index, returns requested SmartRule associated with the Application.
<code>ct_get_smartrules_for_application_by_range</code>	Given an Application, start index, and end index, returns the requested range of SmartRule contained in the Application.
<code>ct_get_smartrules_for_appurl_by_index</code>	Given a ApplicationURL, and an index, returns requested SmartRule associated with the ApplicationURL.
<code>ct_get_smartrules_for_appurl_by_range</code>	Given an ApplicationURL, start index, and end index, returns the requested range of SmartRule contained in the ApplicationURL.
<code>ct_get_userpropdef_for_smartrule</code>	Get the related UserPropertyDefinition for a SmartRule object.
<code>ct_save_smart_rule</code>	Saves the specified SmartRule in the database.

Resources

The following resource objects are provided by the RSA ClearTrust API.

- Application — A logical grouping of resources and functions.
- Application function — A set of functionality within an application.
- URL — A resource labeled by a URI and associated with a particular application and Web server.
- Web server — Associated with URIs, defining the location of the URIs.
- Server tree — Represents a tree of URLs on a Web server

Applications

An *application* object (CT_Application) represents a logical grouping of resources and functions. A user's access to the resources is defined by a basic entitlement from the user to the application.

- If a user is not associated with an application, then accessibility of the contained URLs for the user is determined by how passive access has been defined.
- If a user is associated with an application, the accessibility is defined by whether the association has been defined as accessible.

Table 3.25 CT_Application Object

Element	Type	Description
id	int	Reference for the API layer.
admin_id	int	Reference for the API layer.
admin_name	char *	Name of the owning administrative group.
ct_public	CT_BOOLEAN	Indicates whether the object is viewable by any administrative user.
name	char *	Name of the application, used for object reference and display label on the GUI.
version	char *	Version of the application.
description	char *	Textual description of the application.

The application object contains descriptive information. It does not describe the application's associated with relationships.

The following table describes the `ct_commands.h` functions that operate on an application.

Table 3.26 Functions of `ct_commands.h` Related to Application Objects

Function	Description
<code>ct_create_application</code>	Creates an application object in the database.
<code>ct_delete_application</code>	Deletes the specified application object from the database.
<code>ct_get_app_by_index</code>	Gets the application for the specified index.
<code>ct_get_app_by_name</code>	Gets the application for the specified name.
<code>ct_get_appfunc_for_application_by_index</code>	Gets the ApplicationFunction for the application for the specified index.
<code>ct_get_appfunc_for_application_by_name</code>	Gets the ApplicationFunction for the application for the specified name.
<code>ct_get_appfuncs_for_application_by_names</code>	Gets an array of ApplicationFunctions for the application for the specified names.
<code>ct_get_appfuncs_for_application_by_range</code>	Gets an array of ApplicationFunctions for the application for the specified index range.
<code>ct_get_application_owner</code>	Gets the owner (administrative group) of the application.
<code>ct_get_apps_by_names</code>	Gets an array of applications for the specified names.
<code>ct_get_apps_by_range</code>	Gets an array of applications for the specified index range.
<code>ct_get_appurl_for_application_by_index</code>	Gets the ApplicationURL associated with the application for the specified index.
<code>ct_get_appurls_for_application_by_range</code>	Gets an array of ApplicationURLs associated with the application for the specified index range.
<code>ct_get_exp_entitlement_for_application_by_index</code>	Gets the basic entitlement assigned to the application for the specified index.
<code>ct_get_exp_entitlement_for_application_by_range</code>	Gets an array of basic entitlements assigned to the application for the specified index range.
<code>ct_get_num_of_appfuncs_for_application</code>	Gets the number of ApplicationFunctions associated with the application.
<code>ct_get_num_of_applications</code>	Gets the number of application objects in the database.
<code>ct_get_num_of_appurls_for_application</code>	Gets the number of ApplicationURLs associated with the specified application.
<code>ct_get_num_of_exp_entitlements_for_application</code>	Gets the number of basic entitlements assigned to the application's 'ACCESS' ApplicationFunction.
<code>ct_save_application</code>	Saves the application in the database.
<code>ct_set_application_owner</code>	Sets the owner (administrative group) of the application.

Application Functions

An *application function* (`CT_ApplicationFunction` declared in `ct_structs.h`) represents a protected set of functionality within an application. The set of functionality can be protected through the creation of a basic entitlement.

Table 3.27 `CT_ApplicationFunction` Object

Element	Type	Description
id	int	Reference for the API layer.
name	char *	Name of the ApplicationFunction
description	char *	Textual description of the ApplicationFunction.
filter	CT_BOOLEAN	Filter for conflicting SmartRules. TRUE - "allow" SmartRule takes precedence over a "deny" SmartRule. FALSE - "deny" SmartRule takes precedence over an "allow" SmartRule.

The following table describes the `ct_commands.h` operations that can be performed on an ApplicationFunction object.

Table 3.28 Functions of `ct_commands.h` Related to ApplicationFunctions

Function	Description
<code>ct_create_app_func</code>	Creates an ApplicationFunction object.
<code>ct_delete_app_func</code>	Deletes an ApplicationFunction object.
<code>ct_get_app_for_appfunc</code>	Gets the application related to the ApplicationFunction.
<code>ct_save_appfunc</code>	Saves the ApplicationFunction in the database.

Application URLs

An *ApplicationURL* object (`CT_ApplicationURL`) represents a resource labeled by a URI and associated with a particular application and Web server.

An ApplicationURL represents an accessible URL that is part of a WebApplication.

Table 3.29 `CT_ApplicationURL` Object

Element	Type	Description
id	int	Reference for the API layer.
app_id	int	ID of the URL's application.
websrvr_id	int	ID of the URL's Web server.
description	char *	Textual description of the ApplicationURL.
uri	char *	The accessible file. Note: This does not include the protocol (e.g. http:) or host name.

The ApplicationURL object contains descriptive information. It does not describe the ApplicationURL's associated-with relationships.

The following table describes the `ct_commands.h` functions that operate on an ApplicationURL object.

Table 3.30 ApplicationURL Functions of `ct_commands.h`

Function	Description
<code>ct_create_app_url</code>	Creates an ApplicationURL associated with an Application and Web server.
<code>ct_delete_appurl</code>	Deletes an ApplicationURL from the database.
<code>ct_get_webserver_for_appurl</code>	Gets the Web server associated with the ApplicationURL.
<code>ct_save_appurl</code>	Saves the ApplicationURL in the database.
<code>ct_set_webserver_for_appurl</code>	Sets the Web server associated with the ApplicationURL.

Web Servers

A *Web server* object (CT_WebServer) is associated with URIs, defining the location of the URIs. The Web server also represents the location of the authorizer which performs accessibility checking against the associated URIs.

A Web server defines a URL's location through an associated-with relationship.

The Web server object represents the location of an RSA ClearTrust authorizer which performs accessibility checking against the associated URLs.

Table 3.31 CT_WebServer Object

Element	Type	Description
<code>id</code>	<code>int</code>	Reference for the API layer.
<code>admin_id</code>	<code>int</code>	Reference for the API layer.
<code>admin_name</code>	<code>char *</code>	Name of the owning administrative group.
<code>ct_public</code>	<code>CT_BOOLEAN</code>	Defines the Web server's visibility to RSA ClearTrust.
<code>name</code>	<code>char *</code>	Name of the application. Used for object reference and display label on the GUI.
<code>description</code>	<code>char *</code>	Textual description of the Web server. Used for information only.
<code>manufacturer</code>	<code>char *</code>	Manufacturer of the Web server. Used for information only.
<code>hostname</code>	<code>char *</code>	Host name of the Web server. Used for information only.
<code>port</code>	<code>int</code>	Port number of the Web server. Used for information only.

The following `ct_commands.h` functions operate on a Web server object.

Table 3.32 Web server Functions of `ct_commands.h`

Function	Description
<code>ct_create_web_server</code>	Creates a Web server object in the database.
<code>ct_delete_webserver</code>	Deletes a Web server object from the database.
<code>ct_get_appurl_for_webserver_by_index</code>	Gets the ApplicationURL associated with the Web server for the specified index.
<code>ct_get_appurls_for_webserver_by_range</code>	Gets an array of ApplicationURLs associated with the Web server for the specified index range.
<code>ct_get_num_of_appurls_for_webserver</code>	Gets the number of ApplicationURLs associated with the Web server.
<code>ct_get_num_of_webservers</code>	Gets the number of Web servers in the entitlement server database.
<code>ct_get_webserver_by_index</code>	Gets the Web server for the specified index.
<code>ct_get_webserver_by_name</code>	Gets the Web server for the specified name.
<code>ct_get_webserver_owner</code>	Gets the owner (administrative group) of the Web server.
<code>ct_get_webservers_by_names</code>	Gets an array of Web servers for the specified names.
<code>ct_get_webservers_by_range</code>	Gets an array of Web servers for the specified index range.
<code>ct_save_webserver</code>	Saves the specified Web server in the database.
<code>ct_set_webserver_owner</code>	Sets the owner (administrative group) of the Web server.

Server Trees

A *server tree* object (`CT_ServerTree`) is associated with a Web server. It represents a tree of URLs on a Web server.

Table 3.33 `CT_ServerTree` Object

Element	Type	Description
<code>id</code>	<code>int</code>	Reference for the API layer.
<code>admin_id</code>	<code>int</code>	Reference for the API layer.
<code>admin_name</code>	<code>char *</code>	Name of the owning administrative group.
<code>ct_public</code>	<code>CT_BOOLEAN</code>	Indicates whether the ServerTree is visible to RSA ClearTrust.
<code>uri</code>	<code>char *</code>	Root URI for the ServerTree.
<code>description</code>	<code>char *</code>	Textual description of the ServerTree.

The following `ct_commands.h` functions operate on `ServerTree` objects.

Table 3.34 ServerTree Functions of `ct_commands.h`

Function	Description
<code>ct_create_server_tree</code>	Creates a <code>ServerTree</code> object in the database.
<code>ct_delete_server_tree</code>	Deletes a <code>ServerTree</code> object from the database.
<code>ct_get_num_servertree_for_webserver</code>	Gets the number of <code>ServerTrees</code> associated with the specified Web server.
<code>ct_get_servertree_for_webserver_by_index</code>	Gets the <code>ServerTree</code> associated with the Web server for the specified index.
<code>ct_get_servertree_for_webserver_by_range</code>	Gets an array of <code>ServerTrees</code> associated with the Web server for the specified index range.
<code>ct_get_servertree_owner</code>	Gets the owner (administrative group) of the <code>ServerTree</code> .
<code>ct_save_server_tree</code>	Saves the <code>ServerTree</code> to the database.
<code>ct_set_servertree_owner</code>	Sets the owner (administrative group) of the <code>ServerTree</code> .

Searching

The API provides objects and operations for searching the entitlements database. For detailed function descriptions, see the comments in `<CT_HOME>/api/admin-c/include/ct_search.h`.

Valid search types are:

- `CT_ADMIN_USER_SEARCH`
- `CT_ADMIN_GROUP_SEARCH`
- `CT_GROUP_SEARCH`
- **Deprecated:** `CT_REALM_SEARCH`
- `CT_WEB_SERVER_SEARCH`
- `CT_APPLICATION_SEARCH`
- `CT_USER_PROP_DEF_SEARCH`
- `CT_USER_SEARCH`

Each search type has a corresponding search object to contain search information that is passed to the appropriate search function. The following tables describe the search information for each search object.

Table 3.35 `CT_AdminGroupSearch` Object

Element	Type	Description
<code>nameCriterion</code>	<code>CT_StringCriterion *</code>	Criterion to match the administrative group name.

Table 3.36 CT_GroupSearch Object

Element	Type	Description
nameCriterion	CT_StringCriterion *	Criterion to match the group name.

Table 3.37 Deprecated: CT_RealmSearch Object

Element	Type	Description
nameCriterion	CT_StringCriterion *	Criterion to match the realm name.

Table 3.38 CT_WebServerSearch Object

Element	Type	Description
nameCriterion	CT_StringCriterion *	Criterion to match the Web server name.

Table 3.39 CT_ApplicationSearch Object

Element	Type	Description
nameCriterion	CT_StringCriterion *	Criterion to match the application name.

Table 3.40 CT_UserPropDefSearch Object

Element	Type	Description
nameCriterion	CT_StringCriterion *	Criterion to match the UserPropDef name.

Table 3.41 CT_UserSearch Object

Element	Type	Description
useridCriterion	CT_StringCriterion *	Criterion to match the user ID.
firstNameCriterion	CT_StringCriterion *	Criterion to match the user's first name.
lastNameCriterion	CT_StringCriterion *	Criterion to match the user's last name.
emailCriterion	CT_StringCriterion *	Criterion to match the user's email address.
dnCriterion	CT_StringCriterion *	Criterion to match the user's DN.
accountStartCriterion	CT_DateCriterion *	Criterion to match the user's Account Start Date.
accountEndCriterion	CT_DateCriterion *	Criterion to match the user's Account End Date.
userLockoutCriterion	CT_BooleanCriterion *	Criterion to match the user's lockout attribute.
superUserCriterion	CT_BooleanCriterion *	Criterion to match the user's SuperUser attribute.
superHelpDeskCriterion	CT_BooleanCriterion *	Criterion to match the user's HelpDesk attribute.
ownerCriterion	CT_StringCriterion *	Criterion to match the user's owner's name.
userPropCritArrayRef	CT_UserPropCriteriaArrayRef	A reference to an array of UserPropertyCriteria objects.

Table 3.42 CT_UserInGroupSearch Object

Element	Type	Description
useridCriterion	CT_StringCriterion *	Criterion to match the user ID.
firstNameCriterion	CT_StringCriterion *	Criterion to match the user's first name.
lastNameCriterion	CT_StringCriterion *	Criterion to match the user's last name.
emailCriterion	CT_StringCriterion *	Criterion to match the user's email address.
dnCriterion	CT_StringCriterion *	Criterion to match the user's DN.
accountStartCriterion	CT_DateCriterion *	Criterion to match the user's Account Start Date.
accountEndCriterion	CT_DateCriterion *	Criterion to match the user's Account End Date.
userLockoutCriterion	CT_BooleanCriterion *	Criterion to match the user's lockout attribute.
superUserCriterion	CT_BooleanCriterion *	Criterion to match the user's SuperUser attribute.
superHelpDeskCriterion	CT_BooleanCriterion *	Criterion to match the user's HelpDesk attribute.
ownerCriterion	CT_StringCriterion *	Criterion to match the user's owner's name.
userPropCritArrayRef	CT_UserPropCriteriaArrayRef	A reference to an array of UserPropertyCriteria objects.

To conduct a search, you first must allocate memory for the search Criterion object. You should use the `ct_alloc_search` and `ct_free_search` functions to allocate and free memory for the search objects.

You then must populate the search object with the information required for the search. Once that information is in place, you can pass the search object as a parameter to the appropriate search function.

The following table describes the search functions of `ct_search.h`.

Table 3.43 Search Functions of `ct_search.h`

Function	Description
<code>ct_alloc_search</code>	Allocates a Search object for the specified search type.
<code>ct_free_search</code>	Frees a Search object.
<code>ct_get_admingroup_by_index_in_search</code>	Gets the administrative group that matches the <code>CT_AdminGroupSearch</code> criteria for the specified index.
<code>ct_get_admingroup_by_name_in_search</code>	Gets the administrative group that matches the <code>CT_AdminGroupSearch</code> criteria for the specified name.
<code>ct_get_admingroups_by_names_in_search</code>	Gets an array of administrative groups that match the <code>CT_AdminGroupSearch</code> criteria for the specified names.
<code>ct_get_admingroups_by_range_in_search</code>	Gets an array of administrative groups that match the <code>CT_AdminGroupSearch</code> criteria for the specified index range.
<code>ct_get_app_by_index_in_search</code>	Gets the application that matches the <code>CT_ApplicationSearch</code> criteria for the specified index.
<code>ct_get_app_by_name_in_search</code>	Gets the application that matches the <code>CT_ApplicationSearch</code> criteria for the specified name.

Table 3.43 Search Functions of `ct_search.h`

Function	Description
<code>ct_get_apps_by_names_in_search</code>	Gets an array of applications that match the <code>CT_ApplicationSearch</code> criteria for the specified names.
<code>ct_get_apps_by_range_in_search</code>	Gets an array of applications that match the <code>CT_ApplicationSearch</code> criteria for the specified index range.
<code>ct_get_group_by_index_in_search</code>	Gets the group that matches the <code>CT_GroupSearch</code> criteria for the specified index.
<code>ct_get_group_by_name_in_search</code>	Gets the group that matches the <code>CT_GroupSearch</code> criteria for the specified name.
<code>ct_get_group_by_names_in_search</code>	Gets an array of groups that match the <code>CT_GroupSearch</code> criteria for the specified names.
<code>ct_get_groups_by_range_in_search</code>	Gets an array of groups that match the <code>CT_GroupSearch</code> criteria for the specified index range.
<code>ct_get_num_of_admingroups_in_search</code>	Gets the number of administrative groups that match the <code>CT_AdminGroupSearch</code> criteria.
<code>ct_get_num_of_apps__in_search</code>	Gets the number of applications in the entitlement server database that match the <code>CT_ApplicationSearch</code> criteria.
<code>ct_get_num_of_groups_in_search</code>	Gets the number of groups in the entitlement server database that match the <code>CT_GroupSearch</code> criteria.
<code>ct_get_num_of_realms_in_search</code>	Deprecated: Gets the number of realms in the entitlement server database that match the <code>CT_RealmSearch</code> criteria.
<code>ct_get_num_of_userpropdefs_in_search</code>	Gets the number of <code>UserPropertyDefs</code> in the entitlement server database that match the <code>CT_UserPropDefSearch</code> criteria.
<code>ct_get_num_of_users_in_group_in_search</code>	Gets the number of users in the group that match the <code>CT_UserInGroupSearch</code> criteria.
<code>ct_get_num_of_users_in_search</code>	Gets the number of users in the entitlement server database that match the <code>CT_UserSearch</code> criteria.
<code>ct_get_num_of_webservers_in_search</code>	Gets the number of Web servers in the entitlement server database that match the <code>CT_WebServerSearch</code> criteria.
<code>ct_get_realm_by_index_in_search</code>	Deprecated: Gets the realm that matches the <code>CT_RealmSearch</code> criteria for the specified index.
<code>ct_get_realy_by_name_in_search</code>	Deprecated: Gets the realm that matches the <code>CT_GroupSearch</code> criteria for the specified name.
<code>ct_get_realms_by_names_in_search</code>	Deprecated: Gets an array of realms that match the <code>CT_RealmSearch</code> criteria for the specified names.
<code>ct_get_realms_by_range_in_search</code>	Deprecated: Gets an array of realms that match the <code>CT_RealmSearch</code> criteria for the specified index range.
<code>ct_get_user_by_index_in_search</code>	Gets the user that matches the <code>CT_UserSearch</code> criteria for the specified index.
<code>ct_get_user_by_name_in_search</code>	Gets the user that matches the <code>CT_UserSearch</code> criteria for the specified name.

Table 3.43 Search Functions of `ct_search.h`

Function	Description
<code>ct_get_userpropdef_by_index_in_search</code>	Gets the UserPropDef that matches the CT_UserPropDefSearch criteria for the specified index.
<code>ct_get_userpropdef_by_name_in_search</code>	Gets the UserPropDef that matches the CT_UserPropDefSearch criteria for the specified name.
<code>ct_get_userpropdefs_by_names_in_search</code>	Gets an array of UserPropDefs that match the CT_UserPropDefsSearch criteria for the specified names.
<code>ct_get_userpropdefs_by_range_in_search</code>	Gets an array of UserPropDefs that match the CT_UserPropDefSearch criteria for the specified index range.
<code>ct_get_users_by_names_in_search</code>	Gets an array of user that match the CT_UserSearch criteria for the specified names.
<code>ct_get_users_by_range_in_search</code>	Gets an array of user that match the CT_UserSearch criteria for the specified index range.
<code>ct_get_user_in_group_by_index_in_search</code>	Gets the user in the group that matches the CT_UserInGroupSearch criteria for the specified index.
<code>ct_get_user_in_group_by_name_in_search</code>	Gets the user in the group that matches the CT_UserInGroupSearch criteria for the specified name.
<code>ct_get_users_in_group_by_names_in_search</code>	Gets an array of users in the group that match the CT_UserInGroupSearch criteria for the specified names.
<code>ct_get_users_in_group_by_range_in_search</code>	Gets an array of users in the group that match the CT_UserInGroupSearch criteria for the specified index range.
<code>ct_get_webserver_by_index_in_search</code>	Gets the Web server that matches the CT_WebServerSearch criteria for the specified index.
<code>ct_get_webserver_by_name_in_search</code>	Gets the Web server that matches the CT_WebServerSearch criteria for the specified name.
<code>ct_get_webservers_by_names_in_search</code>	Gets an array of Web servers that match the CT_WebServerSearch criteria for the specified names.
<code>ct_get_webservers_by_range_in_search</code>	Gets an array of Web servers that match the CT_WebServerSearch criteria for the specified index range.

Permissions

The permissions methods let you check whether the administrative user (as logged in under a specific administrative role) has permission to perform a particular action. For detailed function descriptions, see the comments in `<CT_HOME>/api/admin-c/include/ct_permissions.h`.

Table 3.44 Permissions Functions of `ct_permissions.h`

Function	Description
<code>ct_check_add_group_to_realm</code>	Deprecated: Permission to add the specified group to the specified realm.
<code>ct_check_add_user_to_group</code>	Permission to add the specified user to the specified group.
<code>ct_check_change_password</code>	Permission to change the password for the specified user.
<code>ct_check_create_admin_group</code>	Permission to create an administrative group.
<code>ct_check_create_admin_role</code>	Permission to create an administrative role.
<code>ct_check_create_admin_role_in_admin_group</code>	Permission to create an administrative role in the specified administrative group.
<code>ct_check_create_application</code>	Permission to create an application.
<code>ct_check_create_application_function</code>	Permission to create an application function.
<code>ct_check_create_explicit_entitlement</code>	Permission to create a basic entitlement.
<code>ct_check_create_group</code>	Permission to create a group.
<code>ct_check_create_password</code>	Permission to create a password.
<code>ct_check_create_realm</code>	Deprecated: Permission to create a realm.
<code>ct_check_create_server_tree</code>	Permission to create a server tree.
<code>ct_check_create_smart_rule</code>	Permission to create a SmartRule.
<code>ct_check_create_user</code>	Permission to create a user.
<code>ct_check_create_property_definition</code>	Permission to create a user property definition.
<code>ct_check_create_web_server</code>	Permission to create a Web server.
<code>ct_check_delete_administrative_group</code>	Permission to delete the specified administrative group.
<code>ct_check_delete_administrative_role</code>	Permission to delete the specified administrative role.
<code>ct_check_delete_application</code>	Permission to delete the specified application.
<code>ct_check_delete_application_function</code>	Permission to delete the specified application function.
<code>ct_check_delete_explicit_entitlement</code>	Permission to delete the specified basic entitlement.
<code>ct_check_delete_group</code>	Permission to delete the specified group.
<code>ct_check_delete_password_policy</code>	Permission to delete the specified password policy.
<code>ct_check_delete_realm</code>	Deprecated: Permission to delete the specified realm.
<code>ct_check_delete_server_tree</code>	Permission to delete the specified server tree.

Table 3.44 Permissions Functions of `ct_permissions.h`

Function	Description
<code>ct_check_delete_smart_rule</code>	Permission to delete the specified SmartRule.
<code>ct_check_delete_user</code>	Permission to delete the specified user.
<code>ct_check_delete_user_property_definition</code>	Permission to delete the specified user property definition.
<code>ct_check_delete_web_server</code>	Permission to delete the specified Web server.
<code>ct_check_modify_administrative_group</code>	Permission to modify the specified administrative group.
<code>ct_check_modify_administrative_role</code>	Permission to modify the specified administrative role.
<code>ct_check_modify_application</code>	Permission to modify the specified application.
<code>ct_check_modify_application_function</code>	Permission to modify the specified application function.
<code>ct_check_modify_explicit_entitlement</code>	Permission to modify the specified basic entitlement.
<code>ct_check_modify_group</code>	Permission to modify the specified group.
<code>ct_check_modify_password_policy</code>	Permission to modify the specified password policy.
<code>ct_check_modify_realm</code>	Deprecated: Permission to modify the specified realm.
<code>ct_check_modify_server_tree</code>	Permission to modify the specified server tree.
<code>ct_check_modify_smart_rule</code>	Permission to modify the specified SmartRule.
<code>ct_check_modify_user</code>	Permission to modify the specified user.
<code>ct_check_modify_user_property_definition</code>	Permission to modify the specified user property definition.
<code>ct_check_modify_web_server</code>	Permission to modify the specified Web server.
<code>ct_check_set_default_password_policy</code>	Permission to set the default password policy.

Object Utilities

The object utilities provide low level functions that are called by the Administrative C API. For detailed function descriptions, see the comments in `<CT_HOME>/api/admin-c/include/ct_utilities.h`.

Table 3.45 Object Utilities Functions of `ct_utilities.h`

Function	Description
<code>ct_alloc_obj</code>	Allocates a ClearTrust object.
<code>ct_free_obj</code>	Frees an object returned from the API.
<code>ct_alloc_array_obj</code>	Allocates an array of ClearTrust objects.
<code>ct_free_array_obj</code>	Frees an array of objects returned from the API.
<code>ct_free_adminroleid</code>	Frees a <code>CT_AdminRoleId</code> structure and all pointers reachable from the <code>CT_AdminRoleId</code> .
<code>ct_alloc_search</code>	Allocates a Search struct object.
<code>ct_free_search</code>	Frees a Search struct and all pointers reachable from the Search struct.
<code>ct_alloc_criterion</code>	Allocates a Criterion.
<code>ct_free_criterion</code>	Frees a Criterion struct and all pointers reachable from the Criterion.
<code>ct_alloc_userprop_criterion</code>	Allocates a <code>CT_UserPropertyCriterion</code> .
<code>ct_alloc_userprop_criterion_array</code>	Allocates an array of <code>CT_UserPropertyCriterion</code> .
<code>ct_free_userprop_criterion_array</code>	Frees an array of <code>CT_UserPropertyCriterion</code> structs and all pointers reachable from the <code>CT_UserPropertyCriterion</code> array.
<code>ct_alloc_string</code>	Creates an allocated copy of the Source String and returns it through the Destination String.
<code>ct_free_string</code>	Frees an allocated String that was allocated by <code>ct_alloc_string</code> .

Error Codes

This table displays the error code mappings defined in `<CT_HOME>/api/admin-c/include/ct_rc_constants.h`.

Table 3.46 Error Codes Returned by the Administrative C API

Error Code	Numerical Code
<code>RC_OK</code>	0
<code>RC_OBJ_NOT_FOUND</code>	2
<code>RC_INVALID_TYPE</code>	3
<code>RC_NOT_AUTHORIZED</code>	4
<code>RC_TRANSPORT_ERROR</code>	5

Table 3.46 Error Codes Returned by the Administrative C API

Error Code	Numerical Code
RC_MEMORY_ERROR	6
RC_DUPLICATE_OBJ_ERROR	7
RC_RANGE_OUT_OF_BOUNDS	8
RC_NO_AUTH_SERVERS	10
RC_RANGE_EXCEEDS_LIMIT	11
RC_INVALID_REFERENCE	12
RC_BAD_ARGS	13
RC_INVALID_CONTEXT	14
RC_ALREADY_CONNECTED	15
RC_NOT_CONNECTED	16
RC_ALREADY_INITIALIZED	17
RC_ILLEGAL_PASSWORD	18
RC_EXPIRED_PASSWORD	19
RC_AMBIGUOUS_ADMIN_ROLE	20
RC_INCOMPLETE_DATA	21
RC_OPERATION_NOT_APPLICABLE	22
RC_OPERATION_NOT_SUPPORTED	23
RC_SERVER_TIMEOUT	24
RC_INITIALIZATION_ERROR	25
RC_UNKNOWN_ERROR	26

The following error codes have been deprecated as of version 4.7. In the case of a bad password or an invalid administrative role, the API will now return the more general RC_NOT_AUTHORIZED error, instead of those listed below.

Table 3.47 Deprecated Error Codes

Error Code	Numerical Code
RC_INVALID_PASSWORD	1
RC_ADMINISTRATOR_NOT_FOUND	9

Memory Management in the C API

Memory Management when Getting API Objects

When retrieving an object from the server, the RSA ClearTrust C API dynamically allocates the object's storage as well as any associated storage (for example, a string for the object's name, or an array of UserProperties in users).

Freeing Memory

To prevent memory leaks, you must free an object and its associated storage using the `ct_free_obj` function in `ct_utilities.h`.

```
CT_User* user_ptr = NULL;
rc = ct_get_user_and_properties("TestUser", &user_ptr);
..
/* After finished with the CT_User struct you must call
 * ct_free_obj to prevent memory leaks */
rc = ct_free_obj(CT_USER, user_ptr);
```

Freeing Arrays

Similarly, when retrieving an array of objects from the server, the RSA ClearTrust C API dynamically allocates the array of objects as well as their associated storage. To prevent memory leaks, you must free the array of objects through the `ct_free_array_obj` function in `ct_utilities.h`.

```
int num_of_groups = -1;
int ret_array_size = -1;
CT_Group* group_array_ptr = NULL;

rc = ct_get_num_of_groups(&num_of_groups);
if (rc != RC_OK) return rc;
rc = ct_get_groups_by_range(0, num_of_groups-1,
                           &ret_array_size,
                           &group_array_ptr);
if (rc != RC_OK) return rc;
...

/* After finished with the array of CT_Group
 * struct you must call ct_free_array_obj. */
rc = ct_free_array_obj(CT_GROUP, group_array_ptr);
```



Note: DO NOT use the `ct_free_obj` on the individual CT-structs in an array of CT-structs. This will cause unpredictable behavior. The returned array of CT-structs is not an array of pointers to CT-structs.

Memory Management when Modifying an API Object

After retrieving an API object struct, a common task is to make modifications to that struct and then save your changes by passing the struct to the `ct_save` command.

However, you must be careful not to create memory leaks by replacing a reference in the struct without freeing the previous value. Also after any modifications to the struct, when you call any of the `ct_free` functions, the references in that struct must all be NULL or freeable values (i.e., not a `const` value, or a value shared somewhere else).

So, before modifying a struct's attribute, free the attribute's storage and use the appropriate memory utility functions (in `ct_utilities.h`) to replace the attribute with a value that is freeable with the `ct_free` functions.

Example

```
CT_User *user_ptr = NULL;
// Retrieve an object.
rc = ct_get_user_by_name("TestUser", &user_ptr);
if (rc == RC_OK)
{
    // Free the old attribute's Memory
    if (user_ptr->hdr.name != NULL) ct_free_string(
        user_ptr->hdr.name);

    // Don't pass a Constant String to the CT_User
    // struct, so ct_free_obj will be able to
    // deallocate the memory for us.
    ct_alloc_string("ChangedName", user_ptr->hdr.name);

    ct_save_user(user_ptr); //Save changes to the APIServer
    ct_free_obj(CT_USER, user_ptr);
}
```

Memory Management when Creating API Objects

When creating an API object in the RSA ClearTrust Server with the C API, you must pass a struct's reference to the appropriate create function. To avoid memory leaks, this can be accomplished in either of the following two ways:

1. Statically allocate the struct on the stack; or
2. Dynamically allocate the struct with the `ct_alloc` functions.

Static Allocation

This has the benefit that memory management is now left up to the stack and there is no need to call the `ct_free` functions. And since you are not passing the struct to the `ct_free` functions, you can pass string constants to the stack allocated struct. Of course, any memory manually allocated for an attribute of the struct, must be manually freed.

Example:

```

CT_User user;
DATE_TYPE* start_date = localtime(&cur_time);
DATE_TYPE* end_date   = localtime(&cur_time);
end_date->tm_year++;
memset(&user, '\0', sizeof(CT_User));    // Clear out the
                                         // struct's attributes.

/* Set Up the User struct used to Create the User */
user.hdr.name = "JoeBob";
user.hdr.ct_public = TRUE;
user.startdate = *start_date;
user.enddate = *end_date;

rc = ct_create_user_and_properties(&user, 0, NULL);
if (rc != RC_OK)
{
printf ("Error: ct_create_user_and_properties\n");
}

```

However, this has the disadvantage that the struct will only be available for the lifetime of its declared scope. You can retrieve the newly created object's struct with the get functions, but this may be an unwanted and costly overhead.

Dynamic Allocation

Using the `ct_alloc` functions to dynamically allocate the struct to create an object solves the disadvantages of static stack allocation. However, memory management and explicit frees are again left to the programmer.

The `ct_alloc` methods should be used to dynamically allocate the memory of the struct.

Also, it will set all of the allocated struct's attributes references to NULL or 0. To prevent memory leaks the dynamically allocated struct and its associated storage must be freed. When you are done using the Dynamically Allocated struct you must call appropriate `ct_free` function. Again, you must be careful that all of the associated memory is freeable.

Example:

```
CT_User* user_ptr = NULL;
DATE_TYPE *start_date, *end_date;
start_date = localtime(&cur_time);
end_date    = localtime(&cur_time);
end_date->tm_year++;

rc = ct_alloc_obj(CT_USER, &user);
if (rc != RC_OK) return rc;

/* Set Up the User struct used to Create the User */
// First allocate the String so ct_free_obj can
// free the memory for us.
ct_alloc_string("JoeBob", user_ptr->hdr.name);
user_ptr->hdr.ct_public = TRUE;
user_ptr->startdate = *start_date;
user_ptr->enddate = *end_date;

rc = ct_create_user_and_properties(user_ptr, 0, NULL);
if (rc != RC_OK)
{
printf ("Error: ct_create_user_and_properties\n");
}

...
...

/* When done using the user_ptr struct then Free the memory. */
rc = ct_free_obj(CT_USER, user_ptr);
if (rc != RC_OK)
{
printf ("Error: ct_free_obj\n");
}
```

Sample Code

The following sample code illustrates the use of the RSA ClearTrust C Administrative API. For the code to run, your RSA ClearTrust policy datastore must have at least one of each of the following items defined:

- super user
- administrative group
- administrative role

By default, one of each is created when you install RSA ClearTrust.

AdminUser.c

This program shows how to initialize the RSA ClearTrust Administrative API and use it to create, modify, and delete Administrative users. This example is provided in your RSA ClearTrust installation, saved as

<CT_HOME>/api/admin-c/example/AdminUser.c.

```

/* AdminUser.c
 *
 * @version 4.7
 * @since November 20, 2001
 */

#include <time.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <assert.h>

#include "ct_map.h"
#include "ct_search.h"
#include "ct_commands.h"

#define SERVER_NAME      "localhost" /* IP Address may be used 127.0.0.1 */
#define SERVER_PORT     "5601" /* Port Number Entitlement Server is running*/
#define TIMEOUT         300 /* max period of inactivity in seconds */
#define USE_SSL         FALSE /* Either TRUE or FALSE */
#define API_USER        "admin"
#define PASSWORD        "admin1234"

```


AdminUser.c example continues:

```
#define ADMIN_ROLE          "Default Administrative Role"
#define ADMIN_GROUP_NAME   "Default Administrative Group"

int main()
{
    int rc = RC_OK;

    /* Class paths to ClearTrust jar files on your system */
    char *class_path_key =
"c:\\ct\\api\\admin-j\\lib\\ct_admin_api.jar;c:\\ct\\lib\\jcsi_provider.jar;c:\\ct\\lib\\jcel_2-do.jar;c:\\ct\\lib\\jcsi_ssl.jar";

    ct_map* myMap;

    CT_Admin      newAdminRole;
    CT_AdminUser  newAdminUser;

    CT_Admin      *existAdminRole  = NULL;
    CT_AdminUser  *existAdminUser  = NULL;
    CT_AdminGroup *existAdminGroup = NULL;

    time_t cur_time = time(NULL);
    DATE_TYPE* start_date = localtime(&cur_time);
    DATE_TYPE* end_date = localtime(&cur_time);
    end_date->tm_year++;

    /* Set CT_CLASSPATH_KEY to the path to the Java Admin
     * API JAR file (ct_admin_api.jar).
     *
     * You may optionally set CT_DEBUG_KEY to any value
     * to enable debugging and turn off the JIT in the JVM.
     */
    myMap = ct_create_map();
    ct_map_insert( myMap, CT_CLASSPATH_KEY, class_path_key );

    /* Once our ct_map is created and set,
     * we can now initialize the API
     */
    if ((rc = ct_initialize_api( myMap )) != RC_OK) {
        printf("\t[FAIL] ct_initialize_api(): %d\n", rc);
    }
}
```

AdminUser.c example continues:

```

/* Once the API has been initialized,
 * we may destroy the ct_map
 */
ct_map_destroy( myMap );

/* ct_connect() */
if (rc = ct_connect(SERVER_NAME, SERVER_PORT, TIMEOUT,
                   USE_SSL, API_USER, PASSWORD,
                   ADMIN_ROLE, ADMIN_GROUP_NAME )
    != RC_OK) {
    printf("\n\t[FAIL] ct_connect(): %d\n", rc);
} else {
    printf("\n\t[PASS] ct_connect(): %d\n", rc);
}

/* Before an Admin User may be created, an Admin Group
 * and an Admin Role are needed
 */
rc = ct_get_admingroup_by_name("Default Administrative Group",
                              &existAdminGroup);

if ( rc != RC_OK) {
    printf("[FAIL] : ct_get_admingroup_by_name() : %d\n", rc);
} else {

    /* Set a new Admin Role structure*/
    newAdminRole.name           = "Sample AdminRole Name";
    newAdminRole.description    = "Sample AdminRole Description";
    newAdminRole.add_user       = TRUE;
    newAdminRole.mod_user       = TRUE;
    newAdminRole.del_user       = FALSE;
    newAdminRole.add_group      = TRUE;
    newAdminRole.mod_group      = TRUE;
    newAdminRole.del_group      = FALSE;
    newAdminRole.add_realm      = TRUE;
    newAdminRole.mod_realm      = TRUE;
    newAdminRole.del_realm      = FALSE;
}

```

AdminUser.c example continues:

```
newAdminRole.add_app           = TRUE;
newAdminRole.mod_app           = TRUE;
newAdminRole.del_app           = FALSE;
newAdminRole.add_server        = TRUE;
newAdminRole.mod_server        = TRUE;
newAdminRole.del_server        = FALSE;
newAdminRole.add_admin         = TRUE;
newAdminRole.mod_admin         = TRUE;
newAdminRole.del_admin         = FALSE;
newAdminRole.set_password      = TRUE;
newAdminRole.add_user_prop_def = TRUE;
newAdminRole.mod_user_prop_def = TRUE;
newAdminRole.del_user_prop_def = FALSE;

/* Check for duplicate Admin Role and Delete if needed. */
if ((rc = ct_get_adminrole_for_admingroup_by_name(existAdminGroup,
                                                  newAdminRole.name, &existAdminRole)) == RC_OK
    &&
    existAdminRole != NULL
    )
{
    rc = ct_delete_admin_role(existAdminRole);
}

/* Now Create the new Admin Role */
rc = ct_create_admin_role( existAdminGroup, &newAdminRole );
if ( rc != RC_OK) {
    printf("[FAIL] : ct_create_admin_role() : %d\n", rc);
}
}

/* Set up a newAdmin struct */
memset(&newAdminUser, sizeof(CT_AdminUser), 0);

newAdminUser.admin_name      = "TestAdminUser1";
newAdminUser.name            = "TestAdmin1";
```

AdminUser.c example continues:

```

newAdminUser.ct_public           = TRUE;
newAdminUser.password           = "admin1234";
newAdminUser.firstname          = "Test";
newAdminUser.lastname           = "Admin";
newAdminUser.emailaddr          = "tau2@somewhere.com";
newAdminUser.startdate          = *start_date;
newAdminUser.enddate            = *end_date;
newAdminUser.superuser          = FALSE;
newAdminUser.superHelpDesk      = FALSE;
newAdminUser.is_locked          = FALSE;

/* Check for duplicate Admin User and Delete if needed. */
if ((rc = ct_get_admin_user_by_name( newAdminUser.name,
                                     &existAdminUser )) == RC_OK
    &&
    existAdminUser != NULL)
{
    rc = ct_delete_admin_user( existAdminUser );
}

/* Now Create the new Admin Role */
if ((rc = ct_create_admin_user(&newAdminUser, &newAdminRole))
    != RC_OK)
{
    printf("[FAIL] : ct_create_admin_user() : %d\n", rc);
} else {
    printf("\t[PASS] ct_create_admin_user(): %d\n", rc);
}

/*
 * Once we know we have a good Admin User
 * let's go ahead and save changes to it.
 */
{
    CT_AdminUser modifyingAdminUser = newAdminUser;

    modifyingAdminUser.name          = "New name";
    modifyingAdminUser.firstname     = "New first name";

```

AdminUser.c example continues:

```
modifyingAdminUser.lastname    = "New last name";
modifyingAdminUser.emailaddr   = "new_name@somewhere.com";

if ((rc = ct_save_admin_user(&modifyingAdminUser)) != RC_OK)
{
    printf("[FAIL] : ct_save_admin_user() : %d\n", rc);
} else {
    printf("\t[PASS] ct_save_admin_user(): %d\n", rc);
}
}

/* Now let's clean up all admin users and roles */
if ((rc = ct_get_admin_user_by_name( "New name",
                                     &existAdminUser ))
    == RC_OK
    &&
    existAdminUser != NULL
    )
{
    if ((rc = ct_delete_admin_user( existAdminUser )) != RC_OK)
    {
        printf("[FAIL] : ct_delete_admin_user() : %d\n", rc);
    } else {
        printf("\t[PASS] ct_delete_admin_user(): %d\n", rc);
    }
}

if ((rc = ct_delete_admin_role( &newAdminRole )) != RC_OK)
{
    printf("[FAIL] : ct_delete_admin_role() : %d\n", rc);
}

if (rc = ct_disconnect() != RC_OK)
{
    printf("\t[FAIL] ct_disconnect(): %d\n", rc);
} else {
    printf("\t[PASS] ct_disconnect(): %d\n", rc);
}

return rc;
} /* end main() */
```


4

Administrative Java API

This chapter describes the Java version of the RSA ClearTrust Administrative API. The Administrative API allows you to develop security administrator applications that create/update user accounts and set the access rules enforced by the RSA ClearTrust system. The Administrative API uses the Entitlements Server (called the “API Server” when accessed by an API client) to write to the user, policy and administrator data stores on your configured LDAP directory server (or, in future releases, on your relational database management server).

The Administrative API consists of all the classes in the `sirrus.api` packages. See the RSA ClearTrust Administrative API Javadocs for precise descriptions of these classes and methods. You can find the Javadocs in your RSA ClearTrust installation in the directory `<CT_HOME>/api/admin-j/doc/index.html`.

This Chapter

This chapter consists of:

- Compilation instructions starting with “[Installing and Compiling](#)” on page 68.
- Reference information starting with a description of the main Administrative API class, “[APIServerProxy](#)” on page 69.
- Example programs with instructions.
 - “[Connection Example](#)” on page 74
 - “[User Example](#)” on page 110
 - “[User Property Example](#)” on page 113
 - “[Application Function Example](#)” on page 120
 - “[SmartRule Example](#)” on page 123
 - “[User Search Example](#)” on page 127

The source code for these API example programs is installed in

`<CT_HOME>/api/admin-j/example`

Installing and Compiling

This section explains the installed components that make up the API and provides guidelines for building applications. For instructions on installing the APIs, see [Chapter 2, “Installing the RSA ClearTrust APIs”](#).

Compiling Applications

API Library

In order to compile and run API programs, you must have the Administrative API jar, `ct_admin_api.jar`, installed and included in your `SOURCEPATH` and `CLASSPATH`. You will find this jar file installed as:

```
<CT_HOME>/api/admin-j/lib/ct_admin_api.jar
```

SSL Libraries

If your API client program will connect over SSL (see [“Connecting With and Without SSL”](#) on page 72), you will need following additional jar files in your `SOURCEPATH` and `CLASSPATH`. You will find these jars in your RSA ClearTrust installation in the `<CT_HOME>/lib` directory.

RSA SSL software:

- `certj.jar`
- `jsafe.jar`
- `jsafeJCE.jar`
- `rsajsse.jar`
- `sslj.jar`

JCSI Keystore software:

- `jcsi_base.jar`
- `jcsi_provider.jar`

Sun security infrastructure:

- `jce1_2-do.jar`
- `jcet.jar`
- `jnet.jar`
- `jsse.jar`

APIServerProxy

The APIServerProxy provides the communication interface between the client application and the RSA ClearTrust API Server, which is an interface to the Entitlement Server. When using the API, you must first create an APIServerProxy and then connect it to the API Server using the `connect ()` method (see page 72).

APIServerProxy Method Reference

The following table provides an overview of the methods in APIServerProxy. For a detailed description of each method, consult the RSA ClearTrust Javadocs.



Note: You should only connect ONCE to the API Server. If your application is a long running application which issues many requests to the server, connect to the server at the start of your application. If you connect and disconnect for every request to the server, it will drastically degrade the API performance.

Table 4.1 APIServerProxy Methods

Method	Description
APIServerProxy	Constructor to create a server proxy.
checkAccess	Deprecated. Use the Runtime API instead.
checkFunction	Deprecated. Use the Runtime API instead.
checkPassword	Deprecated. Use the Runtime API instead.
connect	Connects the client application to the API Server.
createAdministrativeGroup	Creates a new administrative group.
createAdministrativeUser	Creates a new administrative user.
createApplication	Creates a new application.
createAppURL	Creates a new ApplicationURL.
createExplicitEntitlement	Creates a new basic entitlement, based on the <code>isAccessible</code> switch, between the given entity and an application.
createGroup	Creates a new group.
createPasswordPolicy	Creates a new password policy with parameters matching the default password policy.
createRealm	Deprecated: Creates a new realm.
createUser	Creates a new user.
createUserPropertyDefinition	Creates a new UserPropertyDefinition.

Table 4.1 APIServerProxy Methods

Method	Description
createWebApplication	Deprecated. This has been replaced by the method createApplication().
createWebAppURL	Deprecated. This has been replaced by the method createAppURL().
createWebServer	Creates a new Web server.
disconnect	Disconnects the client application from the server.
flushCache	Forces all the Authorization Servers to flush their caches. You should call flushCache() each time you finish adding or updating users, groups, or entitlements. Note: If you only want to clear the cache of a particular Authorization Server, see clearServerCache() on page 168.
forcePasswordExpiration	Forces the password for the specified user to expire.
getAdministrativeGroup	Returns the administrative group associated with the login session.
getAdministrativeUserByUniqueIdentifier	Get an administrative user by using the unique identifier of the user.
getAdministrativeUsers	Returns a sparse data for all the administrative users in the system.
getAdminGroups	Gets all administrative groups visible to the currently logged-in administrative user.
getAdministrativeRole	Returns the administrative role associated with the login session.
getAdminRoleIdsForUser	Gets an array of AdminRoleIds, which represents all administrative roles that the user currently belongs to.
getApplications	Gets all applications visible to the currently logged-in administrative user.
getAppsForUser	Removed. The getAppsForUser() method in the Java API has been removed. In previous releases, one could get a list of a user's set of available resources by calling this method. This was possible because, in those releases of the product, access to every resource was provided by means of an application function associated with that resource. In version 4.7, access to a given resource may be provided by means of an entitlement to that resource directly, or by means of an entitlement to an application that contains the resource. Since there are multiple ways of providing access to a given resource, the results returned by getAppsForUser() would no longer provide a complete list of accessible resources. For this reason, the method has been removed.
getDefaultPasswordPolicy	Gets the default password policy.
getExplicitEntitlement	Gets a basic entitlement between the given entity and an application.
getGroups	Gets all groups visible to the currently logged-in administrative user.
getPasswordPolicies	Gets all password policies visible to the currently logged-in administrative user.
getPermissionChecker	Gets the permission checker object.
getPWExpirationDate	Gets a user's Password Expiration Date.
getRealms	Deprecated: Gets all realms visible to the currently logged-in administrative user.
getSocket	Gets the socket.
getUser	Fetch the administrative user for the current login session. Returns the logged-in IAdministrativeUser.
getUserAndProperties	Deprecated. All methods that fetch a user now always get the user properties at the same time. See getUsers().

Table 4.1 APIServerProxy Methods

Method	Description
getUserAndPropertiesByDN	Deprecated. All methods that fetch a user always get the user properties at the same time. See getUserByDN(String).
getUserByUniqueIdentifier	This method allows you to retrieve an IUser object that you have previously loaded. Takes an IUniqueIdentifier as an argument. The IUniqueIdentifier allows you to specify which object to load without having to search based on some attribute that may have changed in the meantime, such as the name of the object. You can obtain the UniqueIdentifier by calling getPrimaryKey() on the object that you wish to reload.
getUserPropertyDefinitions	Gets all UserPropertyDefinitions visible to the currently logged-in administrative user.
getUsers	Gets all users visible to the currently logged-in administrative user.
getWebApplications	Deprecated. This method has been replaced by the method getApplications().
getWebServers	Gets all Web servers visible to the currently logged-in administrative user.
isPWExpirationDateOverriden	Determines whether a user's Password Expiration Date overrides the administrative group default password lifetime.
login	Authenticates the user as a ClearTrust administrative user in the API.
resetAdministrativeUserPassword	Resets an administrative user password.
resetPassword	Resets a user's password.
revertAdministrativeUserPasswordExpirationDate	This call reverts the password expiration date for the specified administrative user.
revertPasswordExpirationDate	Reverts the password expiration date for the specified user.
searchAdministrativeGroupObjects	Gets the IAdministrativeGroupSearch object.
searchAdministrativeUserObjects	Gets an IAdministrativeUserSearch object.
searchGroupObjects	Gets an IGroupSearch object.
searchRealmObjects	Deprecated: Gets an IrealmSearch object.
searchUserObjects	Gets an IUserSearch object.
searchUserPropDefObjects	Gets an IUserPropDefSearch object.
searchWebServerObjects	Gets an IWebServerSearch object.
setDefaultPasswordPolicy	Sets the default password policy.
setPassword	Sets a user password.
setPWExpirationDate	Deprecated. Use IUser.setPasswordExpirationDate(Date) to set the user password expiration date.
setTimeout	Sets the timeout value for conversations with the API Server.
validateUser	Deprecated. This method has been replaced by equivalent functionality in the ClearTrust runtime API. See <code>sirus.runtime.RuntimeAPI.authenticate(java.util.Map)</code> . This method is still available in ClearTrust 4.7, but will be removed in a future release.

Connecting an APIServerProxy Client

You will use the `APIServerProxy.connect()` method to connect your API client. Before you begin writing your connection code, you should decide how secure your connection needs to be, as explained in the next section. If you already know what type of SSL or non-SSL connection you will use, turn to [“How to Connect”](#) on page 73 for connection instructions.

Connecting With and Without SSL

An Administrative API client may connect to the Entitlements Server as an authenticated SSL client, an anonymous SSL client, or as a non-SSL client. The `cleartrust.eserver.api_port.use_ssl` setting in the Entitlements Server's `eserver.conf` file indicates which type of connection is required for Administrative API clients. When writing your Administrative API programs, make sure that the boolean you pass as the `use_ssl` parameter of the `APIServerProxy` constructor matches the server's `cleartrust.eserver.api_port.use_ssl` setting. Your settings will match one of the scenarios shown in the subsections that follow.

For more information, See the section “SSL Settings for RSA ClearTrust API Clients” in Chapter 7 of the *RSA ClearTrust Installation and Configuration Guide*.

On a System Running Clear Text Connections

If the RSA ClearTrust system is running with clear text connections between servers, as specified with:

```
cleartrust.net.ssl.use=false
cleartrust.net.ssl.require_authentication=false
```

Then the Administrative API clients must also connect in clear text, as specified with:

```
cleartrust.eserver.api_port.use_ssl=false
```

On a System Running Anonymous SSL Connections

If the RSA ClearTrust system is running with anonymous SSL connections between servers, as specified with:

```
cleartrust.net.ssl.use=true
cleartrust.net.ssl.require_authentication=false
```

Then the Administrative API clients can connect either via clear text:

```
cleartrust.eserver.api_port.use_ssl=false
```

or via anonymous SSL:

```
cleartrust.eserver.api_port.use_ssl=true
```

On a System Running Mutually Authenticated SSL Connections

If the RSA ClearTrust system is running with mutually authenticated SSL connections between servers, as specified with:

```
cleartrust.net.ssl.use=true
```

```
cleartrust.net.ssl.require_authentication=true
```

Then the Administrative API clients can connect either via clear text:

```
cleartrust.eserver.api_port.use_ssl=false
```

or via mutually authenticated SSL:

```
cleartrust.eserver.api_port.use_ssl=true
```

How to Connect

To start an Administrative API session, you must declare an `APIServerProxy` object and then connect it to the API server using the `connect()` method. Enclose your connection code in a try block in order to catch the various exceptions that may be thrown by the `APIServerProxy` constructor and the `connect()` method.

The following example shows just the lines needed to make the connection. Later in this chapter you will see complete example programs that establish a connection.

In this example, we use “localhost” as the name of the API Server machine. If your server is running elsewhere, you must specify the machine name or IP address here.

```
try{
    serverProxy = new APIServerProxy("localhost", //API Server name
                                    5601,       //API Server Port
                                    false);     //Use SSL

    serverProxy.connect("admin",
                       "admin1234",
                       "Default Administrative Group",
                       "Default Administrative Role");
} catch(IOException e){
    System.out.println("\n\n IOError = " + e + "\n\n");
} catch(TransportException e){
    System.out.println("\n\n Error in transport layer = " + e + "\n\n");
} catch(UserNotAuthorizedException e){
    System.out.println("\n\n User unable to act as an Administrator.\n\n");
} catch(APIException e){
    System.out.println("\n\n General API error = " + e + "\n\n");
}
```

Error Messages

If your API client application provides incorrect or insufficient login information to the RSA ClearTrust API Server when calling the `connect()` method, the API will return the following error:

```
sirrus.api.client.UserNotAuthorizedException: Login incorrect
```

This can occur if you passed in an invalid administrative user name, an invalid password, an invalid administrative group, or an invalid administrative role.

Disconnecting an APIServerProxy Client

You can break your connection to the API Server by calling the `disconnect()` method. Generally, your programs should connect only once, calling `disconnect()` only when the program is about to exit. This example assumes you have created an `APIServerProxy` object called “`serverProxy`.”

```

if (serverProxy != null) {
    try {
        serverProxy.disconnect();
    } catch (java.io.IOException e) {
        e.printStackTrace();
    } catch (NotConnectedException e) {
        e.printStackTrace();
    }
}

```

Connection Example

The following example, `AdminUserCheck.java`, connects to the API Server and calls a simple administrative method. The purpose of this example is to demonstrate how to connect and disconnect an Administrative API client.

Edit the Program Before Compiling and Running

Before you can compile and run this example, you may need to make the following edits to the program and installation:

1. Edit the server name and server port parameters of the `APIServerProxy` constructor.
2. Also in the `APIServerProxy` constructor, set the `use_ssl` setting to match the `cleartrust.net.ssl.use` setting in your Entitlements Server's configuration file.
3. Edit the four arguments of the `serverProxy.connect()` method.

For instructions on compiling this example, see [“Compiling Applications”](#) on page 68.

Example

```
package sirrus.samples.admin;

import java.io.*;
import java.util.*;

import sirrus.api.client.*;
import sirrus.api.client.search.*;
import sirrus.api.client.criteria.*;

/**
 * AdminUserCheck.java
 *
 * @version 4.7
 * @since October 19, 2001
 */
public class AdminUserCheck
{
    static APIServerProxy serverProxy = null;
    static String passwordExpiry = new String();

    /**
     *Method to connect to ClearTrust API Server
     */
    public static void connect()
    {
        try{
            serverProxy = new APIServerProxy("localhost", //API Server name
                                           5601,          //API Server Port
                                           false);        //Use SSL

            serverProxy.connect("admin",
                               "admin1234",
                               "Default Administrative Group",
                               "Default Administrative Role");
        }catch(IOException e){
            System.out.println("\n\n IOError = " + e + "\n\n");
        }catch(TransportException e){
            System.out.println("\n\n Error in transport layer = " + e +
"\n\n");
        }catch(UserNotAuthorizedException e){
            System.out.println("\n\n User unable to act as an
Administrator.\n\n");
        }catch(APIException e){
            System.out.println("\n\n General API error = " + e + "\n\n");
        }
    }
}
```

Example continues:

```
/*
 * Method to disconnect from ClearTrust API server
 */
public static void disconnect()
{
    if (serverProxy != null){
        try{
            serverProxy.disconnect();
        }catch(java.io.IOException e){
            e.printStackTrace();
        }catch(NotConnectedException e){
            e.printStackTrace();
        }
    }
}

public static void main(String[] args)
{
    AdminUserCheck apiClient = new AdminUserCheck();

    //Initializing the connection to ClearTrust API server
    apiClient.connect();

    //Get user name
    System.out.println("User name?");
    BufferedReader bfReader =
        new BufferedReader(new InputStreamReader(System.in));
    String str1 = new String("0");
    try{
        str1 = bfReader.readLine();
    }catch(Exception e){
        e.printStackTrace();
    }

    //Check the password expiration date
    System.out.println("Retrieving password expiration date...");

    try{
        passwordExpiry =
            serverProxy.getPWExpirationDate(str1).toString();
    }catch(IOException e){
        e.printStackTrace();
    }catch(TransportException e){
        e.printStackTrace();
    }catch(ObjectNotFoundException e){
        System.out.println("User may not exist.\n");
        e.printStackTrace();
    }
}
```


Example continues:

```
    }catch(APIException e){  
        e.printStackTrace();  
    }  
  
    System.out.println("Password expires: " + passwordExpiry);  
  
    AdminUserCheck.disconnect();  
  
    }  
}
```

Administration Objects

Administrative objects consist of the following:

- Administrative group (IAdministrativeGroup) — Defines which administrative users own (can view and modify) a set of objects.
- Administrative user (IAdministrativeUser) — A user dedicated only to RSA ClearTrust administration activities. An administrative user is not to be confused with an IUser who is granted or denied access to resources that are protected by RSA ClearTrust. IUsers cannot act as administrative users.
- Administrative role (IAdministrator) — A set of permissions defining what an administrative user logged in under this role can and cannot do.
- Password policy (IPasswordPolicy) — A set of restrictions on passwords for users.

The following sections describe the Administration objects.

Administrative Group

The IAdministrativeGroup interface describes the *administrative group*, which is a set of administrative users. An administrative group governs which administrative users can view and modify which entities in the RSA ClearTrust data store. This is done by requiring that each entity (a user, group, realm, application, Web server, server tree, or user property definition) be owned by an administrative group. The administrative users who are members of a given administrative group can view and edit the entities owned by that administrative group. Each administrative user in RSA ClearTrust is a member of one and only one administrative group. See the *RSA ClearTrust Administrator's Guide* for more information on administrative groups and users.

Note that each administrative user's actions are also limited by his administrative role (see page 82).

Finally, the administrative group provides one more thing: the password policy. Each RSA ClearTrust user has a password policy that establishes the rules for his password. The user's password policy is provided by the administrative group that owns that user.

Table 4.2 IAdministrativeGroup Methods

Method	Description
createAdministrator	Creates a new administrative role in this administrative group.
getAdministrators	Gets all administrative users in this administrative group.
getApplications	Gets all applications owned by this administrative group.
getGroups	Gets all groups owned by this administrative group.
getPasswordPolicy	Gets the name of the password policy associated with this administrative group.
getRealms	Deprecated: Gets all realms owned by this administrative group.

Table 4.2 IAdministrativeGroup Methods

Method	Description
getUserPropertyDefinitions	Gets all the user property definitions in this administrative group.
getUsers	Gets all users owned by this administrative group.
getWebServers	Gets all Web servers in this administrative group.
isDefaultPrivate	Checks whether new entities created in this administrative group are private by default.
isForcedPasswordExpiry	Checks whether new users created in this administrative group are forced to change their password on first login.
setDefaultPrivate	Sets whether new entities created in this administrative group are private by default.
setForcedPasswordExpiry	Sets whether new users created in this administrative group are forced to change their password on next login.
setPasswordPolicy	Sets the name of the password policy associated with this administrative group.
transferOwnership	Transfers ownership from the current administrative group to another administrative group.
Inherited from interface <code>sirrus.api.client.IAPIObject</code>	
getPrimaryKey	Gets the primary key of this object.
Inherited from interface <code>sirrus.api.client.INamet</code>	
getName	Gets the name of this object.
setName	Sets the name of this object.
Inherited from interface <code>sirrus.api.client.IDescription</code>	
getDescription	Gets the textual description of this object.
setDescription	Sets the textual description of this object.
Inherited from interface <code>sirrus.api.client.ICreatable</code>	
delete	Deletes this object from the entitlements database.
save	Saves this object to the entitlements database.

Administrative User

Administrative users (IAdministrativeUsers) are users that are used purely for RSA ClearTrust administration activities, and are not users that are granted access to resources that are protected by RSA ClearTrust. In practice, administrative users typically exist entirely within the RSA ClearTrust policy repository (as opposed to a separate user store in LDAP or elsewhere), but this is configurable. This functionality existed on IUser in previous releases, but has since been separated to more easily support read-only user stores.

Administrative users cannot be granted entitlements to RSA ClearTrust-protected resources. If you have an administrative user who wishes to access resources, you must create a separate IUser account for that person. See “Users” on page 89.

An administrative user can edit records owned by his administrative group (see page 78), and his actions are limited to those permitted by his administrative role (see page 82).

Table 4.3 IAdministrativeUser Methods

Method	Description
addAdministrativeRole(IAdministrator role)	Adds the administrative role to this administrative user.
getAdministrativeLockout()	Checks if this administrative user is currently locked out of the system. This is only true if an administrative user has explicitly disabled this administrative user account, and overrides all other account activity settings (start/end date, etc.)
getEmail_address()	Fetches the email address of this administrative user.
getEndDate()	Gets the end date for this administrative user's account.
getFirstName()	Gets the first name of this administrative user.
getLastName()	Gets the last name of this administrative user.
getPasswordExpirationDate()	Gets the password expiration date for this administrative user.
getStartDate()	Gets the start date for this administrative user's account.
isSuperHelpDesk()	Checks if this administrative user is a super help desk user.
isSuperUser()	Checks if this administrative user is a superuser.
removeAdministrativeRole(IAdministrator role)	Removes this administrative user from the specified administrative role.
setAdministrativeLockout(boolean locked)	Disables this administrative user account. Doing this overrides all other account activity settings (start/end date, etc.).
setEmailAddress(String email)	Sets this administrative user's email address.
setEndDate(Date aDate)	Sets the end date for this administrative user's account.
setFirstName(String n)	Sets this administrative user's first name.
setLastName(String n)	Sets the last name of this administrative user.
setPassword(String password)	Sets this administrative user's password.

Table 4.3 IAdministrativeUser Methods

Method	Description
setPasswordExpirationDate(Date date)	Sets the password expiration date for this administrative user.
setStartDate(Date aDate)	Sets the start date for this administrative user's account.
setSuperHelpDesk(boolean isSuperHelpDesk)	Sets this administrative user's super help desk status.
setSuperUser(boolean isSuperUser)	Sets this administrative user's super user status.
Methods inherited from interface <code>sirrus.api.client.IOwnable</code>	
getAdministrativeGroup	Gets the administrative group that owns this administrative user.
getAdministrativeGroupName	Gets the name of the administrative group that owns this administrative user.
isPublic	Gets the public flag. If true, this object is viewable by any administrative user. If false, only members of the owning administrative group can view this object.
setAdministrativeGroup	Sets the administrative group that owns this administrative user.
setPublic	Sets the public flag. If true, this object is viewable by any administrative user. If false, only members of the owning administrative group can view this object.
Methods inherited from interface <code>sirrus.api.client.IName</code>	
getName	Gets the name of this object.
setName	Sets the name of this object.
Methods inherited from interface <code>sirrus.api.client.ICreatable</code>	
delete	Deletes this object from the data store.
save	Saves this object to the data store.
Methods inherited from interface <code>sirrus.api.client.IAPIObject</code>	
getUniquelIdentifier	Returns an opaque token that can be used later to retrieve this object.

Administrative Role

An IAdministrator object is an *administrative role*. An administrative user is assigned one or more administrative roles, which determine which actions he can perform in the RSA ClearTrust system. Administrative roles are usually named to reflect the real-world role of the administrative user (for example, help desk or human resources). The administrative role controls what actions the administrative user can perform on the objects owned by his administrative group.



Note: Do not be confused by the interface name IAdministrator. This interface is not used to model administrative users; administrative users are modeled as IAdministrativeUsers.

The following table describes the functions that operate on the administrative role object.

Table 4.4 Administrative Role (IAdministrator) Methods

Method	Description
addUser	Deprecated. Replaced with addAdministrativeUser().
addAdministrativeUser	Add an administrative user to this administrative role.
getAdministrativeUsers	Gets all administrative users associated with this role.
getCreateAdministrativeRole	Checks whether administrative users in this role can create new administrative roles in this administrative group.
getCreateAdministrativeUser	Checks whether administrative users in this role can create new administrative users owned by this administrative group.
getCreateApplication	Checks whether administrative users in this role can create new applications owned by this administrative group.
getCreateGroup	Checks whether administrative users in this role can create new groups owned by this administrative group.
getCreateRealm	Deprecated: Checks whether administrative users in this role can create new realms owned by this administrative group.
getCreateServer	Checks whether administrative users in this role can create new Web servers owned by this administrative group.
getCreateUser	Checks whether administrative users in this role can create new users owned by this administrative group.
getCreateUserPropertyDefinition	Checks whether administrative users in this role can create new user property definitions owned by this administrative group.
getDeleteAdministrativeRole	Checks whether administrative users in this role can delete existing administrative roles in this administrative group.
getDeleteAdministrativeUser	Checks whether administrative users in this role can delete existing administrative users owned by this administrative group.

Table 4.4 Administrative Role (IAdministrator) Methods

Method	Description
getDeleteApplication	Checks whether administrative users in this role can delete existing applications owned by this administrative group.
getDeleteGroup	Checks whether administrative users in this role can delete existing groups owned by this administrative group.
getDeleteRealm	Deprecated: Checks whether administrative users in this role can delete existing realms owned by this administrative group.
getDeleteServer	Checks whether administrative users in this role can delete existing Web servers owned by this administrative group.
getDeleteUser	Checks whether administrative users in this role can delete existing users owned by this administrative group.
getDeleteUserPropertyDefinition	Checks whether administrative users in this role can delete existing user property definitions owned by this administrative group.
getModifyAdministrativeRole	Checks whether administrative users in this role can modify existing administrative roles in this administrative group.
getModifyAdministrativeUser	Checks whether administrative users in this role can modify existing administrative users owned by this administrative group.
getModifyApplication	Checks whether administrative users in this role can modify existing applications owned by this administrative group.
getModifyGroup	Checks whether administrative users in this role can modify existing groups owned by this administrative group.
getModifyRealm	Deprecated: Checks whether administrative users in this role can modify existing realms owned by this administrative group.
getModifyServer	Checks whether administrative users in this role can modify existing Web servers owned by this administrative group.
getModifyUser	Checks whether administrative users in this role can modify existing users owned by this administrative group.
getModifyUserPropertyDefinition	Checks whether administrative users in this role can modify existing user property definitions owned by this administrative group.
getResetPassword	Checks whether administrative users in this role can reset the password for this administrative group.
getUsers	Deprecated. Replaced with getAdministrativeUsers().
removeUser	Deprecated. Replaced with removeAdministrativeUser().
removeAdministrativeUser	Removes an administrative user from this role.
setCreateAdministrativeRole	Sets whether administrative users in this role can create new administrative roles in this administrative group.
setCreateAdministrativeUser	Sets whether administrative users in this role can create new administrative users in this administrative group.
setCreateApplication	Sets whether administrative users in this role can create new applications in this administrative group.
setCreateGroup	Sets whether administrative users in this role can create new groups in this administrative group.

Table 4.4 Administrative Role (IAAdministrator) Methods

Method	Description
setCreateRealm	Deprecated: Sets whether administrative users in this role can create new realms in this administrative group.
setCreateServer	Sets whether administrative users in this role can create new Web servers in this administrative group.
setCreateUser	Sets whether administrative users in this role can create new users in this administrative group.
setCreateUserPropertyDefinition	Sets whether administrative users in this role can create new user property definitions in this administrative group.
setDeleteAdministrativeRole	Sets whether administrative users in this role can delete existing administrative roles in this administrative group.
setDeleteAdministrativeUser	Sets whether administrative users in this role can delete existing administrative users owned by this administrative group.
setDeleteApplication	Sets whether administrative users in this role can delete existing applications owned by this administrative group.
setDeleteGroup	Sets whether administrative users in this role can delete existing groups owned by this administrative group.
setDeleteRealm	Deprecated: Sets whether administrative users in this role can delete existing realms owned by this administrative group.
setDeleteServer	Sets whether administrative users in this role can delete existing Web servers owned by this administrative group.
setDeleteUser	Sets whether administrative users in this role can delete existing users owned by this administrative group.
setDeleteUserPropertyDefinition	Sets whether administrative users in this role can delete existing user property definitions owned by this administrative group.
setModifyAdministrativeRole	Sets whether administrative users in this role can modify existing administrative roles in this administrative group.
setModifyAdministrativeUser	Sets whether administrative users in this role can modify existing administrative users in this administrative group.
setModifyApplication	Sets whether administrative users in this role can modify existing applications in this administrative group.
setModifyGroup	Sets whether administrative users in this role can modify existing groups in this administrative group.
setModifyRealm	Deprecated: Sets whether administrative users in this role can modify existing realms in this administrative group.
setModifyServer	Sets whether administrative users in this role can modify existing Web servers in this administrative group.
setModifyUser	Sets whether administrative users in this role can modify existing users in this administrative group.
setModifyUserPropertyDefinition	Sets whether administrative users in this role can modify existing user property definitions in this administrative group.

Table 4.4 Administrative Role (IAdministrator) Methods

Method	Description
setResetPassword	Sets whether administrative users in this role can modify existing Passwords in this administrative group.
Inherited from interface <code>sirrus.api.client.IAPIObject</code>	
getPrimaryKey	Gets the primary key of this object.
Inherited from interface <code>sirrus.api.client.INamet</code>	
getName	Gets the name of this object.
setName	Sets the name of this object.
Inherited from interface <code>sirrus.api.client.IDescription</code>	
getDescription	Gets the textual description of this object.
setDescription	Sets the textual description of this object.
Inherited from interface <code>sirrus.api.client.ICreatable</code>	
delete	Deletes this object from the entitlements database.
save	Saves this object to the entitlements database.

Password Policy

A *password policy* is a set of restrictions on passwords for users. Each administrative group has an associated password policy that is applied to users owned by that administrative group.

Table 4.5 Password Policy Methods

Method	Description
getDictionaryFile	Gets the name of the dictionary file containing passwords excluded by this policy.
getExclusionCharacters	Gets the list of characters excluded from passwords that satisfy this policy.
getForceNonLetter	Checks whether a password must contain a non-letter character to satisfy this policy.
getHistorySize	Gets the number of past passwords that cannot be reused.
getPasswordLifetime	Gets the lifetime of passwords defined under this password policy.
getPasswordMaximumLength	Gets the maximum allowed length for passwords under this policy.
getPasswordMinimumLength	Gets the minimum allowed length for passwords under this policy.
setDictionaryFile	Sets the dictionary file containing passwords excluded by this policy.
setExclusionCharacters	Sets the list of characters excluded from passwords that satisfy this policy.
setForceNonLetter	Sets whether a password must contain a non-letter character to satisfy this policy.
setHistorySize	Sets the number of past passwords that cannot be reused.
setLengthParams	Sets the range of password lengths allowed under this policy.
setPasswordLifetime	Sets the lifetime of passwords defined under this password policy.

Table 4.5 Password Policy Methods

Method	Description
Inherited from interface <code>sirrus.api.client.IAPIObject</code>	
<code>getPrimaryKey</code>	Gets the primary key of this object.
Inherited from interface <code>sirrus.api.client.INamet</code>	
<code>getName</code>	Gets the name of this object.
<code>setName</code>	Sets the name of this object.
Inherited from interface <code>sirrus.api.client.IDescription</code>	
<code>getDescription</code>	Gets the textual description of this object.
<code>setDescription</code>	Sets the textual description of this object.
Inherited from interface <code>sirrus.api.client.ICreatable</code>	
<code>delete</code>	Deletes this object from the entitlements database.
<code>save</code>	Saves this object to the entitlements database.

Participants

Participant objects model the people and organizations whose access to resources is governed by the RSA ClearTrust system.

- Group: a collection of users
- User: a person who will, upon successful authentication and authorization, be given access to RSA ClearTrust-protected resources.
- User property: an extra detail about a user that can be used as a criterion for access decisions, for Web personalization, etc. A user property is stored in a field that has been declared and defined in a user property definition.
- User property definition: Mechanism for adding extra data fields to user records. In order to add a data field that is usable on all user records, you must create a user property definition that establishes the name and datatype. Once you have created and saved the user property definition, you can begin storing data in the new user property field.



Warning: When creating users and groups, please note that their names may not contain any of the following characters: “, ”, “+”, “””, “\”, “<”, “>” or “;”.

Groups

An IGroup object is a *group*, which is a collection of users and/or other groups. Any user or group can be included in many groups.



Note: If your installation uses collections of groups, please note that the mechanism for doing this has changed with the release of RSA ClearTrust 4.7. Previously, RSA ClearTrust provided the IRealm object for building collections of groups. In version 4.7, the IRealm interface is deprecated, and this functionality has been replaced with nested groups. This means that a group may contain other groups, which may in turn contain other groups, and so on. The deprecation of IRealm means that, while groups may still be collected into realms in 4.7, the IRealm interface will no longer exist in the next version of RSA ClearTrust.

Table 4.6 Group Methods

Method	Description
getCreationDate	Allows entities to query the date a group was created in the Entitlements Database.
Inherited from interface <code>sirrus.api.client.IAPIObject</code>	
getPrimaryKey	Gets the primary key of this object.
Inherited from interface <code>sirrus.api.client.INamet</code>	
getName	Gets the name of this object.
setName	Sets the name of this object. When creating a group, please note that its name may not contain any of the following characters: “;”, “+”, “””, “\”, “<”, “>” or “.”.
Inherited from interface <code>sirrus.api.client.ICreatable</code>	
delete	Deletes this object from the entitlements database.
save	Saves this object to the entitlements database.
Inherited from interface <code>sirrus.api.client.IOwnable</code>	
getAdministrativeGroup	Gets the administrative group that will own the ownable object.
getAdministrativeGroupName	Gets the name of the administrative group that the ownable object is in.
isPublic	Determines whether the object is viewable by any administrative user.
setAdministrativeGroup	Sets the administrative group that will own the ownable object.
setPublic	Sets the public flag.
Inherited from interface <code>sirrus.api.client.IEntity</code>	
createExplicitEntitlement	Creates a new basic entitlement for this entity, the specified application, and the specified application function.
getExplicitEntitlement	Gets this entity's basic entitlement for the specified application and application function.
getExplicitEntitlements	Gets all the basic entitlements for the entity.
Inherited from interface <code>sirrus.api.client.IChild</code>	
getParents	Gets all the parents for this child.
Inherited from interface <code>sirrus.api.client.IParent</code>	
addChild	Adds a child to the container.
getChildren	Gets all the children within the parent.
removeChild	Removes a child from the container.

Users

An IUser object is a *user* who will attempt to view or use an RSA ClearTrust-protected URL or other resource. Users are usually collected into groups and are given rights to resources via basic entitlements and SmartRules. A user may be a member of many groups.



Warning: Do not confuse users with administrative users; they are separate and unrelated objects. See [“Administrative User”](#) on page 80.

The following table describes the methods of IUser.

Table 4.7 User Methods

Method	Description
getCreationDate	Allows entities to query the date a group was created in the Entitlements Database.
getDN	Gets the Distinguished Name for the user.
getEmailAddress	Gets the user's email address.
getEndDate	Gets the end date for the user's account.
getFirstName	Gets the user's first name.
getLastName	Gets the user's last name.
getStartDate	Gets the start date for the user's account.
getUserProperties	Gets all the user properties associated with the user.
getUserProperty	Gets a Property for a user.
isAdminLockedout	Determines whether the user is locked out of the system.
isSuperHelpDesk	Determines whether the user is a super helpdesk user.
isSuperUser	Determines whether the user is a superuser.
setAdminLockedout	Sets whether the user is locked out of the system.
setDN	Sets the Distinguished Name for the user. When creating a user, please note that its name may not contain any of the following characters: “ ”, “+”, “””, “\”, “<”, “>” or “,”.
setEmailAddress	Sets the user's email address.
setEndDate	Sets the end date for the user's account.
setFirstName	Sets the user's first name.
setLastName	Sets the user's last name.
setPassword	Sets the user's password.
setStartDate	Sets the start date for the user's account.
setSuperHelpDesk	Sets the user's super helpdesk status.
setSuperUser	Sets the user's super user status.

Table 4.7 User Methods

Method	Description
setUserProperty	Sets a user property for a user.
Inherited from interface <code>sirrus.api.client.IAPIObject</code>	
getPrimaryKey	Gets the primary key of this object.
Inherited from interface <code>sirrus.api.client.INamet</code>	
getName	Gets the name of this object.
setName	Sets the name of this object. When creating a user, please note that its name may not contain any of the following characters: “, “+”, “””, “\”, “<”, “>” or “;”.
Inherited from interface <code>sirrus.api.client.ICreatable</code>	
delete	Deletes this object from the entitlements database.
save	Saves this object to the entitlements database.
Inherited from interface <code>sirrus.api.client.IOwnable</code>	
getAdministrativeGroup	Gets the administrative group that will own the ownable object.
getAdministrativeGroupName	Gets the name of the administrative group that the ownable object is in.
isPublic	Determines whether the object is viewable by any administrative user.
setAdministrativeGroup	Sets the administrative group that will own the ownable object.
setPublic	Sets the public flag.
Inherited from interface <code>sirrus.api.client.IEntity</code>	
createExplicitEntitlement	Creates a new basic entitlement for this entity, the specified application, and the specified application function.
getExplicitEntitlement	Gets this entity's basic entitlement for the specified application and application function.
getExplicitEntitlements	Gets all the basic entitlements for the entity.
Inherited from interface <code>sirrus.api.client.IChild</code>	
getParents	Gets all the parents for this child.

User Properties

An IUserProperty object is a *user property*, which is an extra detail about a user that can be used as a criterion for access decisions, for Web personalization, etc. A user property is stored in a field that has been declared and defined in a user property definition (see “[User Property Definitions](#)” below). User property values are used by SmartRules to determine authorizations for users.

Table 4.8 User Property Methods

Method	Description
getUser	Gets the user associated with this user property.
getValue	Gets the value for this user property.
getValueType	Gets the type of the user property.
isSet	Determines whether this user property has been set.
setValue	Sets the user property value as an object.
Inherited from interface <code>sirrus.api.client.IAPIObject</code>	
getPrimaryKey	Gets the primary key of this object.
Inherited from interface <code>sirrus.api.client.INamet</code>	
getName	Gets the name of this object.
Inherited from interface <code>sirrus.api.client.ICreatable</code>	
delete	Deletes the object from the entitlements database.
save	Saves the object to the entitlements database.

User Property Definitions

An IUserPropertyDefinition object is a *user property definition*, which establishes an extra data field on user records. In order to add a data field that is usable on all user records, you must create a user property definition that establishes the name and datatype of the user property. Once you have created and saved the user property definition, you can begin storing data in the new user property.

Table 4.9 UserPropertyDefinition Methods

Method	Description
getValueType	Gets the type of this user property definition.
setValueType	Sets the type of this user property definition.
isExportable	Returns a boolean indicating whether or not UserProperties associated with this definition are visible to Runtime API clients.
setExportable	Sets the boolean indicating whether or not UserProperties associated with this definition will be visible to Runtime API clients.

Table 4.9 UserPropertyDefinition Methods

Method	Description
isHelpDeskAccessible	Determines whether the user property definition can be seen by the super helpdesk administrative user.
setHelpDeskAccessible	Sets whether the user property definition can be seen by the super helpdesk administrative user.
isReadOnly	Determines whether the user property definition is read only.
setReadOnly	Sets whether the user property definition is read only.
Inherited from interface <code>sirrus.api.client.IAPIObject</code>	
getPrimaryKey	Gets the primary key of this object.
Inherited from interface <code>sirrus.api.client.INamet</code>	
getName	Gets the name of this object.
setName	Sets the name of this object.
Inherited from interface <code>sirrus.api.client.IDescription</code>	
getDescription	Gets the textual description of this object.
setDescription	Sets the textual description of this object.
Inherited from interface <code>sirrus.api.client.ICreatable</code>	
delete	Deletes this object from the entitlements database.
save	Saves this object to the entitlements database.
Inherited from interface <code>sirrus.api.client.IOwnable</code>	
getAdministrativeGroup	Gets the administrative group that will own the ownable object.
setAdministrativeGroup	Sets the administrative group that will own the ownable object.
getAdministrativeGroupName	Gets the name of the administrative group that the ownable object is in.
isPublic	Returns the boolean indicating whether this object is viewable by any administrative user.
setPublic	Determines whether this object is viewable by any administrative user.

Deprecated Interface: IRealm

IRealm is a deprecated interface for building collections of groups. As of 4.7, groups may be collected together in other groups, so realms are no longer needed. See “Groups” on page 87.

Table 4.10 Methods of the deprecated IRealm interface

Method	Description
getCreationDate	Allows entities to query the date a realm was created in the Entitlements Database.
Inherited from interface <code>sirrus.api.client.IAPIObject</code>	
getPrimaryKey	Gets the primary key of this object.
Inherited from interface <code>sirrus.api.client.INamet</code>	
getName	Gets the name of this object.
setName	Sets the name of this object.
Inherited from interface <code>sirrus.api.client.ICreatable</code>	
delete	Deletes this object from the entitlements database.
save	Saves this object to the entitlements database.
Inherited from interface <code>sirrus.api.client.IOwnable</code>	
getAdministrativeGroup	Gets the administrative group that will own the ownable object.
getAdministrativeGroupName	Gets the name of the administrative group that the ownable object is in.
isPublic	Determines whether the object is viewable by any administrative user.
setAdministrativeGroup	Sets the administrative group that will own the ownable object.
setPublic	Sets the public flag.
Inherited from interface <code>sirrus.api.client.IEntity</code>	
createExplicitEntitlement	Creates a new basic entitlement for this entity, the specified application, and the specified application function.
getExplicitEntitlement	Gets this entity's basic entitlement for the specified application and application function.
getExplicitEntitlements	Gets all the basic entitlements for the entity.
Inherited from interface <code>sirrus.api.client.IParent</code>	
addChild	Adds a child to the container.
getChildren	Gets all the children within the parent.
removeChild	Removes a child from the container.

Policy Objects

Policy objects describe entitlements and rules that allow participants access to resources.

- Entitlements — Governs a user's access to an application function based on the user's name or his or her membership in a group.
- SmartRules — Governs a user's access to an application function based on his or her user properties.

Basic Entitlements (Explicit Entitlements)

The `IExplicitEntitlement` interface defines an explicit entitlement, which is usually called a *basic entitlement* in the RSA ClearTrust system. A basic entitlement defines a user's or group's access to an application.

In order to create an Entitlement, you must specify its Entity ID and its application ID.

An Entitlement specifically grants or denies a user or group permission to access an application. Basic entitlements granted to a group apply to all users in that group. Basic entitlements have a hierarchy; an entitlement granted or denied at the user level overrides any conflicting entitlements at the group level. Likewise, an entitlement granted or denied at the group level overrides any conflicting entitlement set on a parent group of that group.

In other words, when using nested group structures, if a user has an entitlement assigned at the level of a group and also has a conflicting entitlement assigned at the level of a second group that is higher up the group/parent group hierarchy, then the entitlement of the lowest-level group will take precedence. The reason for this is that the lowest-level group is thought of as providing the most precise definition of the user's privileges.

For example, if group `BronzeUsers` is denied access to application `CarLoanCalculator`, and the `BronzeUsers` group in turn contains a number of groups, then all the users in all the sub-groups of `BronzeUsers` are denied access. If you wished to override this denial for some users, there are two ways you could do it. One approach would be to create a new group (say, group `BronzeSpecial`) for those users, make that group a member of the `BronzeUsers` group, and give that group access to `CarLoanCalculator`. The `BronzeSpecial` setting will override the `BronzeUsers` setting. The other approach would be to grant access at the individual user level. For example, if you grant user `TBradshaw` access to application `CarLoanCalculator`, this overrides any denials for `TBradshaw` defined in the groups to which he belongs.

Table 4.11 Explicit Entitlement (Basic Entitlement) Methods

Method	Description
getApplicationFunction	Gets the application function related to this basic entitlement.
getEntity	Gets the Entity related to this basic entitlement.
isAccessible	Determines whether the Entitlement allows the Entity access to an application.
setAccessible	Sets the accessibility of the Entity to an application.
Inherited from interface <code>sirrus.api.client.IAPIObject</code>	
getPrimaryKey	Gets the primary key of this object.
Inherited from interface <code>sirrus.api.client.ICreatable</code>	
delete	Deletes this object from the entitlements database.
save	Saves this object to the entitlements database.

SmartRules

The `ISmartRule` interface defines a *SmartRule*. A `SmartRule` is associated with a user property definition and an application function.

Table 4.12 SmartRule Methods

Method	Description
getApplicationFunction	Gets the application function associated with this <code>SmartRule</code> .
getCategory	Gets the category set in this <code>SmartRule</code> .
getSmartRuleCriteria	Gets the criteria set in this <code>SmartRule</code> .
getUserPropertyDefinition	Gets the user property definition associated with this <code>SmartRule</code> .
setCategory	Sets the category for this <code>SmartRule</code> .
setSmartRuleCriteria	Sets the criteria for this <code>SmartRule</code> .
Inherited from interface <code>sirrus.api.client.IAPIObject</code>	
getPrimaryKey	Gets the primary key of this object.
Inherited from interface <code>sirrus.api.client.ICreatable</code>	
delete	Deletes this object from the entitlements database.
save	Saves this object to the entitlements database.

Resources

The following resource objects are provided by the RSA ClearTrust API.

- Application — A logical grouping of resources and functions.
- Application function — A set of functionality within an application.
- URL — A resource labeled by a URI and associated with a particular application and Web server.
- Web server — Associated with URIs, defining the location of the URIs.
- Server Tree — Represents a tree of URLs on a Web server

Applications

The IApplication interface defines an *application*.

An application consists of a collection of URLs specifying resources and a set of application functions that delineate the possible ways in which that collection of resources can be accessed or manipulated.

A user's access to the resources is defined by a basic entitlement from the user to the application.

- If a user is not associated with an application, then accessibility of the contained URLs for the user is determined by how passive access has been defined.
- If a user is associated with an application, the accessibility is defined by whether the association has been defined as accessible.

Table 4.13 IApplication Methods

Function	Description
createApplicationFunction	Creates a new application function.
createApplicationURL	Creates an ApplicationURL and associates it with this application.
createSmartRule	Creates a SmartRule associated with this application. See the code example on page 123 .
getApplicationFunctions	Gets all the application functions associated with this application.
getApplicationURLs	Gets all the application's URLs.
getVersion	Gets the version of this application.
setVersion	Sets the version of this application.
Inherited from interface <code>sirrus.api.client.IAPIObject</code>	
getPrimaryKey	Gets the primary key of this object.
Inherited from interface <code>sirrus.api.client.IName</code>	
getName	Gets the name of this object.

Table 4.13 IApplication Methods

Function	Description
setName	Sets the name of this object.
Inherited from interface <code>sirrus.api.client.IDescription</code>	
getDescription	Gets the textual description of this object.
setDescription	Sets the textual description of this object.
Inherited from interface <code>sirrus.api.client.ICreatable</code>	
delete	Deletes this object from the entitlements database.
save	Saves this object to the entitlements database.
Inherited from interface <code>sirrus.api.client.IOwnable</code>	
getAdministrativeGroup	Gets the administrative group that will own the ownable object.
getAdministrativeGroupName	Gets the name of the administrative group that the ownable object is in.
isPublic	Determines whether the object is viewable by any administrative user.
setAdministrativeGroup	Sets the administrative group that will own the ownable object.
setPublic	Sets the public flag.

Application Functions

The `IApplicationFunction` interface defines an *application function*.

An application function is a ClearTrust representation of any type of function or method in any type of custom application that you might build. Modeling a method as an application function allows RSA ClearTrust to control access to that method. This allows you to implement ClearTrust Agent-like controls (similar to building a ClearTrust WAX) governing access to methods in your custom applications.

For example, if you are creating a non-Web Java application that has a sensitive method that you do not wish to make available to all users, you can govern access to this method by creating an `ApplicationFunction` for it. For a method called `updateBalance()`, for example, you could create an `ApplicationFunction` record called “update_balance” in the RSA ClearTrust policy datastore. The implementation of your `updateBalance()` method will call `RuntimeAPI.authorize()` before executing the rest of the method, passing as arguments a `Map` of the requesting user and a `Map` representing the “update_balance” method. The `authorize()` call returns a boolean indicating whether the user is allowed to execute the method.

When setting policies in the RSA ClearTrust system, you treat an `ApplicationFunction` like you would treat an `ApplicationURL`. That is, you collect related `ApplicationFunctions` (along with any related `ApplicationURLs`) into applications, and you grant users or groups permission to use these applications, `ApplicationFunctions`, and/or `ApplicationURLs`.

If you are not building stand-alone applications, then you will generally not need to use `ApplicationFunctions`. For most dynamic Web content such as cgis on Web pages,

you will control access by creating an ApplicationURL that matches the URL request string or the name of the cgi script being called. Once you have created Entitlements based on these ApplicationURLs, the RSA ClearTrust Web Server Agents can control access to the functions in the cgi script. In contrast, ApplicationFunctions are useful for access checking in situations that do not involve URL requests, that is, in situations where the RSA ClearTrust Agents cannot be used.

Table 4.14 IApplicationFunction Methods

Method	Description
createSmartRule	Creates a SmartRule associated with this application function. In previous versions, in order to create any SmartRule whatsoever, you had to use this method (<code>IApplicationFunction.createSmartRule</code>). Now, for most SmartRules, you will instead use the <code>IApplicationURL.createSmartRule</code> or <code>IApplication.createSmartRule</code> method. See the code example on page 123 .
getApplication	Gets the application associated with this application function.
getExplicitEntitlements	Gets the basic entitlements associated with this application function
getSmartRules	Gets the SmartRules associated with this application function.
isPolicyAllowBeforeDeny	Returns the policy precedence for this application function. True means the ALLOW policy overrides the DENY policy if there is a conflict.
setPolicyAllowBeforeDeny	Sets the policy precedence for this application function. True means the ALLOW policy overrides the DENY policy if there is a conflict.
Inherited from interface <code>sirrus.api.client.IAPIObject</code>	
getUniqueId	Returns the store-independent unique identifier of the object.
Inherited from interface <code>sirrus.api.client.INamet</code>	
getName	Gets the name of this object.
setName	Sets the name of this object.
Inherited from interface <code>sirrus.api.client.IDescription</code>	
getDescription	Gets the textual description of this object.
setDescription	Sets the textual description of this object.
Inherited from interface <code>sirrus.api.client.ICreatable</code>	
delete	Deletes this object from the entitlements database.
save	Saves this object to the entitlements database.

Application URLs

The IApplicationURL interface defines the *ApplicationURL* object.

An ApplicationURL object represents a resource labeled by a URI and associated with a particular application and Web server. An ApplicationURL represents an accessible URL that is part of a WebApplication.

Table 4.15 ApplicationURL Methods

Method	Description
createSmartRule	Creates a SmartRule associated with this URI. See the code example on page 123 .
getApplication	Gets the application that is identified by this URI.
getURI	Gets the value of this URI.
getWebServer	Gets the Web server that is associated with this URI.
setURI	Sets the value of this URI.
setWebServer	Sets the Web server that is associated with this IRI.
Inherited from interface <code>sirrus.api.client.IAPIObject</code>	
getPrimaryKey	Gets the primary key of this object.
Inherited from interface <code>sirrus.api.client.IDescription</code>	
getDescription	Gets the textual description of this object.
setDescription	Sets the textual description of this object.
Inherited from interface <code>sirrus.api.client.ICreatable</code>	
delete	Deletes this object from the entitlements database.
save	Saves this object to the entitlements database.

Web Servers

The IWebServer interface defines a virtual *Web server* (potentially a group of many physical Web servers) relative to which these URIs are defined.

A Web server object is associated with URIs, defining the location of the URIs. The Web server also represents the location of the authorizer which performs accessibility checking against the associated URIs.

A Web server defines a URL's location through an associated-with relationship.

The Web server object represents the location of an RSA ClearTrust authorizer which performs accessibility checking against the associated URLs.

Table 4.16 Web Server Methods

Method	Description
createServerTree	Creates a new server tree on this Web server.
getHostname	Gets the hostname of the computer on which the Web server is running.
getManufacturer	Gets the maker of the Web server. Valid values are: APACHE MICROSOFT NETSCAPE_ENTERPRISE NETSCAPE_FASTTRACK
getPort	Gets the port number to which the Web server is listening.
getServerTrees	Gets all the Sever Trees that are associated with this Web server.
getWebApplicationURLs	Gets all the Web application URLs associated with this Web server.
setHostname	Sets the hostname of the computer on which the Web server is running.
setManufacturer	Sets the maker of the Web server. Valid values are: APACHE MICROSOFT NETSCAPE_ENTERPRISE NETSCAPE_FASTTRACK
setPort	Sets the Port Number to which the Web server is listening.
Inherited from interface <code>sirrus.api.client.IAPIObject</code>	
getPrimaryKey	Gets the primary key of this object.
Inherited from interface <code>sirrus.api.client.INamet</code>	
getName	Gets the name of this object.
setName	Sets the name of this object.
Inherited from interface <code>sirrus.api.client.IDescription</code>	
getDescription	Gets the textual description of this object.
setDescription	Sets the textual description of this object.
Inherited from interface <code>sirrus.api.client.ICreatable</code>	
delete	Deletes this object from the entitlements database.
save	Saves this object to the entitlements database.
Inherited from interface <code>sirrus.api.client.IOwnable</code>	
getAdministrativeGroup	Gets the administrative group that will own the ownable object.
getAdministrativeGroupName	Gets the name of the administrative group that the ownable object is in.
isPublic	Determines whether the object is viewable by any administrative user.
setAdministrativeGroup	Sets the administrative group that will own the ownable object.
setPublic	Sets the public flag.

Server Trees

The IServerTree interface defines a *server tree*.

A ServerTree object is associated with a Web server object. It represents a tree of URLs on a Web server.

Table 4.17 IServerTree Methods

Method	Description
getURI	Gets the URI for this server tree.
setURI	Sets the URI for this server tree.
Inherited from interface <code>sirrus.api.client.IAPIObject</code>	
getPrimaryKey	Gets the primary key of this object.
Inherited from interface <code>sirrus.api.client.INamet</code>	
getName	Gets the name of this object.
setName	Sets the name of this object.
Inherited from interface <code>sirrus.api.client.IDescription</code>	
getDescription	Gets the textual description of this object.
setDescription	Sets the textual description of this object.
Inherited from interface <code>sirrus.api.client.ICreatable</code>	
delete	Deletes this object from the entitlements database.
save	Saves this object to the entitlements database.
Inherited from interface <code>sirrus.api.client.IOwnable</code>	
getAdministrativeGroup	Gets the administrative group that will own the ownable object.
getAdministrativeGroupName	Gets the name of the administrative group that the ownable object is in.
isPublic	Determines whether the object is viewable by any administrative user.
setAdministrativeGroup	Sets the administrative group that will own the ownable object.
setPublic	Sets the public flag.

Utility Classes

ISparseData

A set of RSA ClearTrust objects is typically loaded as an ISparseData object, generally called a Sparse Data object. A Sparse Data holds a homogenous set of objects and provides methods for retrieving individual objects or subsets of objects.

Sparse data objects are usable only as iterators over the data set. Due to the on-demand availability of data in the physical data store, it is not possible to accurately fetch the size of a data set without first retrieving the entire set. Applications should therefore treat ISparseData objects as iterators. The correct way to iterate over them is shown in the following example:

```
try
{
    int i = 0;
    while (true)
    {
        Object obj = sparseData.getByIndex(i++);
        // The sparse data is of size i or greater, process the object
        ....
    }
}
catch (ObjectNotFoundException onf)
{
    // We have read all of the entries in the result set,
    // continue execution
}
```

See the [EditPropertyExample](#) on page 115 for a more detailed example of using Sparse Data objects.

Table 4.18 ISparseData Methods

Function	Description
getByIndex(int index)	Returns the object at the specified index.
getByName(java.lang.String name)	Returns the object in the ISparseData collection with the specified name.
getByNames(java.lang.String[] names)	Returns a subset of the ISparseData collection as an array of IAPIObjets.
getByRange(int startRange, int endRange)	Returns a subset of the ISparseData collection as an array of IAPIObjets.

Permissions

The `IAdministrativePermissionChecker` interface provides a set of operations that determine whether the administrative role under which the API user is logged in has permission to perform that action.

Table 4.19 IAdministrativePermissions Methods

Function	Description
<code>checkAddGroupToRealm</code>	Deprecated: Permission to add the specified group to the specified realm.
<code>checkAddUserToGroup</code>	Permission to add the specified user to the specified group.
<code>checkChangePassword</code>	Permission to change the password for the specified user.
<code>checkCreateAdministrativeGroup</code>	Permission to create an administrative group.
<code>checkCreateAdministrativeRole</code>	Permission to create an administrative role.
<code>checkCreateAdministrativeRole</code>	Permission to create an administrative role in the specified administrative group.
<code>checkCreateApplication</code>	Permission to create an application.
<code>checkCreateApplicationFunction</code>	Permission to create an application function.
<code>checkCreateExplicitEntitlement</code>	Permission to create a basic entitlement.
<code>checkCreateGroup</code>	Permission to create a group.
<code>checkCreatePasswordPolicy</code>	Permission to create a password.
<code>checkCreateRealm</code>	Deprecated: Permission to create a realm.
<code>checkCreateServerTree</code>	Permission to create a server tree.
<code>checkCreateSmartRule</code>	Permission to create a SmartRule.
<code>checkCreateUser</code>	Permission to create a user.
<code>checkCreateUserPropertyDefinition</code>	Permission to create a Property Definition.
<code>checkCreateWebServer</code>	Permission to create a Web server.
<code>checkDeleteAdministrativeGroup</code>	Permission to delete the specified administrative group.
<code>checkDeleteAdministrativeRole</code>	Permission to delete the specified administrative role.
<code>checkDeleteApplication</code>	Permission to delete the specified application.
<code>checkDeleteApplicationFunction</code>	Permission to delete the specified application function.
<code>checkDeleteExplicitEntitlement</code>	Permission to delete the specified basic entitlement.
<code>checkDeleteGroup</code>	Permission to delete the specified group.
<code>checkDeletePasswordPolicy</code>	Permission to delete the specified password policy.
<code>checkDeleteRealm</code>	Deprecated: Permission to delete the specified realm.
<code>checkDeleteServerTree</code>	Permission to delete the specified server tree.
<code>checkDeleteSmartRule</code>	Permission to delete the specified SmartRule.
<code>checkDeleteUser</code>	Permission to delete the specified user.

Table 4.19 IAdministrativePermissions Methods

Function	Description
checkDeleteUserPropertyDefinition	Permission to delete the specified Property Definition.
checkDeleteWebServer	Permission to delete the specified Web server.
checkModifyAdministrativeGroup	Permission to modify the specified administrative group.
checkModifyAdministrativeRole	Permission to modify the specified administrative role.
checkModifyApplication	Permission to modify the specified application.
checkModifyApplicationFunction	Permission to modify the specified application function.
checkModifyExplicitEntitlement	Permission to modify the specified basic entitlement.
checkModifyGroup	Permission to modify the specified group.
checkModifyPasswordPolicy	Permission to modify the specified password policy.
checkModifyRealm	Deprecated: Permission to modify the specified realm.
checkModifyServerTree	Permission to modify the specified server tree.
checkModifySmartRule	Permission to modify the specified SmartRule.
checkModifyUser	Permission to modify the specified user.
checkModifyUserPropertyDefinition	Permission to modify the specified Property Definition.
checkModifyWebServer	Permission to modify the specified Web server.
checkModifyPasswordPolicy	Permission to set the default password policy.
checkRemoveUserFromGroup	Checks whether the user's administrative role has permission to remove the specified user from a group.
checkRemoveGroupFromRealm	Deprecated: Checks whether the user's administrative role has permission to remove the specified group from a realm.
checkViewPasswordPolicy	Checks whether the user has permission to view the password policies.

Criteria

The criteria classes specify rules for matching a particular property type.

Boolean Criterion

The BooleanCriterion class specifies a rule for matching a Boolean Property.

Table 4.20 BooleanCriterion Methods

Method	Description
BooleanCriterion	BooleanCriterion constructor.
getBooleanValue	Gets the Boolean value.
getObjectType	Gets the criterion's object type.
getOperator	Gets the Boolean operator.
readObject	Reads the criterion from the inStream.
setBooleanValue	Sets the Boolean value.
setOperator	Sets the boolean operator.
writeObject	Writes the criterion to the outStream.

Date Criterion

The DateCriterion class specifies a rule for matching a Date property.

Table 4.21 DateCriterion Methods

Method	Description
DateCriterion	DateCriterion constructor.
getDateValue	Gets the Date value.
getObjectType	Gets the criterion's object type.
getOperator	Gets the Date operator.
readObject	Reads the criterion from the inStream.
setDateValue	Sets the Date value.
setOperator	Sets the Date operator.
writeObject	Writes the criterion to the outStream.

Float Criterion

The FloatCriterion class specifies a rule for matching a Float property.

Table 4.22 FloatCriterion Methods

Method	Description
FloatCriterion	FloatCriterion constructor.
getFloatValue	Gets the Float value.
getObjectType	Gets the criterion object type.
readObject	Reads the criterion from the inStream.
setFloatValue	Sets the Float value.
writeObject	Writes the criterion to the outStream.
Inherited from interface <code>sirrus.api.client.INameSearch</code>	
getOperator	Gets the number operator.
setOperator	Sets the number operator.

Integer Criterion

The IntegerCriterion class specifies a rule for matching an Integer property.

Table 4.23 IntegerCriterion Methods

Method	Description
IntegerCriterion	IntegerCriterion constructor.
getIntegerValue	Gets the Integer value.
getObjectType	Gets the criterion object type.
readObject	Reads the criterion from the inStream.
setIntegerValue	Sets the Integer value.
writeObject	Writes the criterion to the outStream.
Inherited from interface <code>sirrus.api.client.INameSearch</code>	
getOperator	Gets the number operator.
setOperator	Sets the number operator.

Searching

Searching for an object consists of specifying the proper criteria and invoking the appropriate Search interface. The Criteria classes specify the rules for the various criteria needed for searches. The Search interfaces perform the search.

Administrative Group Search

The IAdministrativeGroupSearch interface allows users to specify criteria to search for IAdministrativeGroups.

Table 4.24 IAdministrativeGroupSearch Methods

Method	Description
Inherited from interface <code>sirrus.api.client.search.IAPIObjectSearch</code>	
<code>search</code>	Gets a SparseData of APIObjects, based on the current state of this Search object.
Inherited from interface <code>sirrus.api.client.INameSearch</code>	
<code>getNameCriterion</code>	Gets the name criterion.
<code>setNameCriterion</code>	Sets the name criterion.

Application Search

The IApplicationSearch interface allows users to specify criteria to search for IApplications.

Table 4.25 IApplicationsSearch Methods

Method	Description
Inherited from interface <code>sirrus.api.client.search.IAPIObjectSearch</code>	
<code>search</code>	Gets a SparseData of APIObjects, based on the current state of this Search object.
Inherited from interface <code>sirrus.api.client.INameSearch</code>	
<code>getNameCriterion</code>	Gets the name criterion.
<code>setNameCriterion</code>	Sets the name criterion.

Group Search

The IGroupSearch interface allows users to specify criteria to search for IGroups.

Table 4.26 IGroupSearch Methods

Method	Description
Inherited from interface <code>sirrus.api.client.search.IAPIObjectSearch</code>	
<code>search</code>	Gets a SparseData of APIObjects, based on the current state of this Search object.
Inherited from interface <code>sirrus.api.client.INameSearch</code>	
<code>getNameCriterion</code>	Gets the name criterion.
<code>setNameCriterion</code>	Sets the name criterion.

Deprecated: Realm Search

The deprecated IRealmSearch interface allows users to specify criteria to search for IRealms. This interface is deprecated as of RSA ClearTrust 4.7 because realms will soon be removed from the system.

Table 4.27 IRealmSearch Methods

Method	Description
Inherited from interface <code>sirrus.api.client.search.IAPIObjectSearch</code>	
<code>search</code>	Gets a SparseData of APIObjects, based on the current state of this Search object.
Inherited from interface <code>sirrus.api.client.INameSearch</code>	
<code>getNameCriterion</code>	Gets the name criterion.
<code>setNameCriterion</code>	Sets the name criterion.

User Property Definition Search

The IUserPropDefSearch interface allows users to specify criteria to search for IUserPropertyDefinitions.

Table 4.28 IUserPropDefSearch Methods

Method	Description
Inherited from interface <code>sirrus.api.client.search.IAPIObjectSearch</code>	
<code>search</code>	Gets a SparseData of APIObjects, based on the current state of this Search object.
Inherited from interface <code>sirrus.api.client.INameSearch</code>	
<code>getNameCriterion</code>	Gets the name criterion.
<code>setNameCriterion</code>	Sets the name criterion.

Web Server Search

The IWebServerSearch interface allows users to specify criteria to search for IWebServers.

Table 4.29 IWebServerSearch Methods

Method	Description
Inherited from interface <code>sirrus.api.client.search.IAPIObjectSearch</code>	
<code>search</code>	Gets a SparseData of APIObjects, based on the current state of this Search object.
Inherited from interface <code>sirrus.api.client.INameSearch</code>	
<code>getNameCriterion</code>	Gets the name criterion.
<code>setNameCriterion</code>	Sets the name criterion.

User Search

The IUserSearch interface allows users to specify criteria to search for IUsers.

Table 4.30 IUserSearch Methods

Method	Description
clearUserPropertyCriterion	Clears or removes the UserPropertyCriterion based on the UserProperty's name.
getAccountEndCriterion	Gets the account end date criterion.
getAccountStartCriterion	Gets the account start date criterion.
getAllUserPropertyCriteria	Retrieve an enumeration of all the UserPropertyCriteria that are currently set.
getDNCriterion	Gets the Distinguished Name criterion.
getEmailAddressCriterion	Gets the email address criterion.
getFirstNameCriterion	Gets the first name criterion.
getLastNameCriterion	Gets the last name criterion.
getOwnerCriterion	Gets the owner criterion.
getSuperHelpDeskCriterion	Gets the SuperHelpDesk criterion.
getSuperUserCriterion	Gets the SuperUser criterion.
getUserIDCriterion	Gets the user ID criterion.
getUserLockoutCriterion	Gets the user lockout criterion.
lookupUserPropertyCriterion	Looks up a UserPropertyCriterion based on the UserProperty name.
numOfUserPropertyCriteria	Gets the number of UserPropertyCriteria set in this search.
putUserPropertyCriterion	Adds a UserPropertyCriterion to this UserSearch.
setAccountEndCriterion	Sets the account end date.
setAccountStartCriterion	Sets the account start date.
setDNCriterion	Sets the Distinguish Name criterion.
setEmailAddressCriterion	Sets the email address criterion.
setFirstNameCriterion	Sets the first name criterion.
setLastNameCriterion	Sets the last name criterion.
setOwnerCriterion	Sets the owner criterion.
setSuperHelpDeskCriterion	Sets the SuperHelpDesk criterion.
setSuperUserCriterion	Sets the SuperUser criterion.
setUserIDCriterion	Sets the user ID criterion.
setUserLockoutCriterion	Sets the user lockout criterion.

Examples

User Example

This Administrative API example, `UserExample.java`, creates a user, saves the user to the database, modifies that user, and finally deletes it.

Before you can compile and run this program you should make sure that a user property exists whose name matches the name used in the `setUserProperty()` call below. You may need to expose an LDAP attribute in the RSA ClearTrust system as a user property and/or edit the `UserProperty` name used in this program. See the Developer's Guide for information on exposing attributes as user properties.

Before You Compile and Run the Program

Before you can compile and run this example, you may need to make the following edits to the program and to your installation:

1. Edit the server name, server port, and `use_ssl` parameters of the `APIServerProxy` constructor. The server name is the name of the machine where the Entitlements Server is running. The port is the API server port on the Entitlements Server. By default this is 5601, and it is generally not changed from the default.
2. Set the `use_ssl` setting to match the `cleartrust.eserver.api.ssl.use` or `cleartrust.net.ssl.use` setting in your Entitlements Server's `eserver.conf` file. See [Connecting With and Without SSL](#) on page 3-1 for details.

```
APIServerProxy myServerProxy =
    new APIServerProxy("localhost" //API Server name
                      ,5601        //API Server Port
                      ,false       //Use SSL
                      );
```

3. Edit the four arguments of the `serverProxy.connect()` method. The four arguments are the administrative user name, his/her password, the administrative group to which the administrative user belongs, and his/her administrative role. The values shown below are the defaults.

```
myServerProxy.connect("admin",
                      "adminpassword",
                      "Default Administrative Group",
                      "Default Administrative Role");
```

For instructions on compiling this example, see [“Compiling Applications”](#) on page 68.

Example: UserExample.java

```
package sirrus.samples.admin;

import java.util.Date;
import sirrus.api.client.*;

/**
 * UserExample.java
 *
 * @version 4.7
 * @since October 23, 2001
 */
public class UserExample {
    public static void main (String[] args)
    {
        try
        {
            APIServerProxy myServerProxy =
                new APIServerProxy("localhost" //API Server name
                                   ,5601      //API Server Port
                                   ,false     //Use SSL
                                   );

            // Connect to the API Server Proxy.
            myServerProxy.connect("admin",
                                  "admin1234",
                                  "Default Administrative Group",
                                  "Default Administrative Role");
            System.out.println ("Connected.");

            // Create a User Object.
            String userID = "testuser767";
            boolean isPublic = true;
            Date startDate = new Date();
            Date endDate = new Date();
            String firstName = "Joe";
            String lastName = "Robertsham";
            String emailAddr = "joebob@bigco.com";
            String password = "Joes1InitialPassword";
            boolean isSuperUser = false;
            IUser user = myServerProxy.createUser(userID,
                                                    isPublic,
                                                    startDate,
                                                    endDate,
                                                    firstName,
                                                    lastName,
                                                    emailAddr,
                                                    password,
                                                    isSuperUser);

            // Save the new user in the Entitlements database.
            user.save();
            System.out.println ("User " + user.getName() + " saved.");
        }
    }
}
```

Example continues:

```

// Modify the User's UserProperty.
// Note: This LDAP attribute must be exposed as a
// CT UserProperty before you run this program.
// See the comment at the beginning of this file.
user.setUserProperty("employeeType",
                    "Outside Sales");

user.save();
System.out.println ("User " + user.getName() + " modified.");

System.out.println ("Flushing cache.");
// Flush the cache so that all Auth Servers will see the new
// user. Note! You should call the flushCache() method only
// infrequently; overusing it can slow system performance.
// This is because flushCache() tells all the Auth Servers
// to reload their cached data.
myServerProxy.flushCache();
System.out.println ("Cache flush complete.");

// Display the UserProperties.
System.out.println ("Retrieving user properties...");
IUserProperty stringProp =
    user.getUserProperty("employeeType");
System.out.println ("For user " + user.getName()
                  + ", the value of " + stringProp.getName()
                  + " is " + stringProp.getValue() + ".");

// Remove the user from the Entitlements database.
user.delete();

// Call flushCache to make the Auth Servers aware of
// the deletion.
myServerProxy.flushCache();
System.out.println ("User deleted.");

    }
    catch (Exception e)
    {
        e.printStackTrace();
    }
}
}

```

User Property Example

EditPropertyExample shows how to load a user and work with his or her UserProperties. RSA ClearTrust may be installed with or without write access to the LDAP user datastore. The `userToEdit.save()` line in this program will only work if you have write access.

Before you compile and run this example, you should do the following:

1. Edit the server name, server port, and `use_ssl` parameters of the `APIServerProxy` constructor below.
2. Set the `use_ssl` setting to match the `cleartrust.eserver.api.ssl.use` or `cleartrust.net.ssl.use` setting in your Entitlements Server's `eserver.conf` file.
3. Edit the four arguments of the `serverProxy.connect()` method.
4. In your LDAP user datastore, make sure that you have saved at least one user to work with.
5. In your LDAP user datastore, make sure that one or two attributes are exposed on the user records you will use.
6. In the RSA ClearTrust Entitlements Manager, make sure you have created a UserProperty Definition for each such exposed LDAP user attribute. This example works with String properties.

For instructions on compiling this example, see [“Compiling Applications”](#) on page 68.

Example: EditPropertyExample.java

```
package sirrus.samples.admin;

import java.io.*;
import java.util.*;
import sirrus.api.client.*;

/**
 * EditPropertyExample.java
 *
 * @version 4.7
 * @since October 22, 2001
 */
public class EditPropertyExample
{
    APIServerProxy serverProxy = null;

    /**
     *Method to connect to ClearTrust API Server
     */
    private void connect()
    {
        try{
            serverProxy = new APIServerProxy("localhost" //API Server name
                                           ,5601 //API Server Port
                                           ,false //Use SSL
                                           );

            serverProxy.connect("admin",
                               "admin1234",
                               "Default Administrative Group",
                               "Default Administrative Role");
        }catch(IOException e){
            System.out.println("\n\n IOError = " + e + "\n\n");
        }catch(TransportException e){
            System.out.println("\n\n Error in transport layer = " + e + "\n\n");
        }catch(UserNotAuthorizedException e){
            System.out.println("\n\n User unable to act as an Administrator.\n\n");
        }catch(APIException e){
            System.out.println("\n\n General API error = " + e + "\n\n");
        }
    }
}
```

Example continues:

```
/*
 * Flushes the caches.
 */
private void flushCache()
{
    if (serverProxy != null){
        try{
            serverProxy.flushCache();
        }catch(java.io.IOException e){
            e.printStackTrace();
        }catch(NoAuthorizersAvailableException e){
            e.printStackTrace();
        }catch(TransportException e){
            e.printStackTrace();
        }
    }
}

/*
 * Method to disconnect from ClearTrust API server
 */
private void disconnect()
{
    if (serverProxy != null){
        try{
            serverProxy.disconnect();
        }catch(java.io.IOException e){
            e.printStackTrace();
        }catch(NotConnectedException e){
            e.printStackTrace();
        }
    }
}

/*
 * Loads a user
 */
private IUser loadUser(String userIdentifier)
{
    ISparseData userSparse = null;
    IUser loadedUser = null;

    try{
        userSparse = serverProxy.getUsers();
        loadedUser = (IUser) userSparse.getBy_name(userIdentifier);
        System.out.println("Successfully loaded the user " +
            loadedUser.getName());
    }catch(ObjectNotFoundException e){
        System.out.println("Cannot find a user called " +
            userIdentifier + "\n\n");
    }
}
```

Example continues:

```

    }catch(APIException e){
        e.printStackTrace();
    }catch(java.io.IOException e){
        e.printStackTrace();
    }

    return loadedUser;
}

/*
 * Prints a user's properties.
 */
private void printUserProps(IUser userToPrint)
{
    // Retrieve a SparseData
    ISparseData propsSparseData = userToPrint.getUserProperties();

    // Find out how many items are in the SparseData
    int itemCount = 0;
    try{
        while (true)
        {
            Object obj =
                propsSparseData.getByIndex(itemCount++);
        }
    }catch(ObjectNotFoundException onf){
        System.out.println ("Number of UserProperties: " + (itemCount-1));
    }catch(BadArgumentException e){
        e.printStackTrace();
    }catch(TransportException e){
        e.printStackTrace();
    }catch(java.io.IOException e){
        e.printStackTrace();
    }

    // Get the objects by range and print them.
    try{
        IAPIObject[] propArray =
            propsSparseData.getByRange(0, itemCount-1);
        for (int i = 0; i < propArray.length; i++)
        {
            IUserProperty returnedProp = (IUserProperty)propArray[i];
            if(returnedProp.getName() != null &&
                returnedProp.getValue() != null)
            {
                System.out.println ("Property (" +
                    returnedProp.getName() +
                    ") has a value of " +
                    returnedProp.getValue().toString());
            }
        }
    }
    }catch(BadArgumentException e){
        e.printStackTrace();
    }
}

```


Example continues:

```

    }catch(TransportException e){
        e.printStackTrace();
    }catch(java.io.IOException e){
        e.printStackTrace();
    }
}

/**
 * Changes the value of a String user property.
 */
private void setStringUserProp(IUser userToEdit,
                               String propName,
                               String propValue)
{
    // Retrieve a SparseData
    ISparseData propsSparseData = userToEdit.getUserProperties();

    // Iterates to find the property, then sets it
    // and saves the user.
    int itemCount = 0;
    try{
        while (true)
        {
            IUserProperty returnedProp =
                (IUserProperty) propsSparseData.getByIndex(itemCount);
            if((propName.compareTo(returnedProp.getName()))==0)
            {
                returnedProp.setValue(propValue);
                userToEdit.setUserProperty(returnedProp.getName(),
                                           returnedProp.getValue());
                System.out.println("New value " + propValue +
                                   " was set for Property (" +
                                   returnedProp.getName() + ")");
                userToEdit.save();
                System.out.println("Saved");
                return;
            }
            itemCount++;
        }
    }catch(ObjectNotFoundException e){
        System.out.println ("Did not find a property called " + propName);
        e.printStackTrace();
    }catch(InvalidTypeException e){
        System.out.println ("Wrong type for property " + propName);
        e.printStackTrace();
    }catch(BadArgumentException e){
        e.printStackTrace();
    }catch(TransportException e){
        e.printStackTrace();
    }
}

```

Example continues:

```
    }catch(APIException e){
        e.printStackTrace();
    }catch(java.io.IOException e){
        e.printStackTrace();
    }
}

public static void main(String[] args)
{
    EditPropertyExample apiClient = new EditPropertyExample();

    //Initializing the connection to ClearTrust API server
    apiClient.connect();

    //Get user name
    System.out.println("User name?");
    BufferedReader bfReader =
        new BufferedReader(new InputStreamReader(System.in));
    String userNameIn = new String("0");
    try{
        userNameIn = bfReader.readLine();
    }catch(Exception e){
        e.printStackTrace();
    }

    //Get prop name
    System.out.println("Name of UserProperty to edit?");
    bfReader = new BufferedReader(new InputStreamReader(System.in));
    String propNameIn = new String("0");
    try{
        propNameIn = bfReader.readLine();
    }catch(Exception e){
        e.printStackTrace();
    }

    //Get prop value
    System.out.println("New value for UserProperty?");
    bfReader = new BufferedReader(new InputStreamReader(System.in));
    String propValueIn = new String("0");
    try{
        propValueIn = bfReader.readLine();
    }catch(Exception e){
        e.printStackTrace();
    }
}
```

Example continues:

```
System.out.println("\n");
IUser userToInspect = apiClient.loadUser(userNameIn);
apiClient.printUserProps(userToInspect);
apiClient.setStringUserProp(userToInspect,
                             propNameIn,
                             propValueIn);

apiClient.flushCache();
apiClient.disconnect();
}
}
```

Application Function Example

You can enforce access control on any method of any application that you write by modeling that method as an application function (IApplicationFunction). With your method saved in the RSA ClearTrust policy datastore as an application function, your application can make a Runtime API call to check a user's access to that method.

For instructions on compiling examples, see ["Compiling Applications"](#) on page 68.

```
package sirrus.samples.admin;

import java.util.Date;
import sirrus.api.client.*;

/**
 * ApplicationFunctionExample.java
 *
 * @version 4.7
 * @since October 19, 2001
 */
public class ApplicationFunctionExample {
    public static void main (String[] args)
    {
        try
        {
            APIServerProxy myServerProxy =
                new APIServerProxy("localhost" //API Server name
                                   ,5601      //API Server Port
                                   ,false     //Use SSL
                                   );

            // Connect to the APIServerProxy.
            myServerProxy.connect("admin",
                                  "admin1234",
                                  "Default Administrative Group",
                                  "Default Administrative Role");
            System.out.println ("Connected");

            // Create an Application.
            IApplication app =
                myServerProxy.createApplication(
                                   "TestApplication",
                                   "App Description",
                                   "Ver2");

            app.save();
            System.out.println ("Application created");

            // Create an ApplicationFunction based on the
            // newly created Application.
            IApplicationFunction appFunc =
                app.createApplicationFunction("MyAppFunc",
                                             "AppFunc Description");
        }
    }
}
```

Example continues:

```
appFunc.save();
System.out.println ("ApplicationFunction created");

// Retrieve a SparseData of the ApplicationFunctions.
ISparseData appFuncSparseData =
    app.getApplicationFunctions();

// Find out how many ApplicationFunctions there are.
int numOfAppFuncs = 0;
try
{
    while (true)
    {
        Object obj =
            appFuncSparseData.getByIndex (numOfAppFuncs++);
    }
}
catch (ObjectNotFoundException onf)
{
    System.out.println ("numOfAppFuncs: " + (numOfAppFuncs-1));
}

// Get the objects by range.
IAPIObject [] appFuncArray =
    appFuncSparseData.getByRange(0, numOfAppFuncs-1);
for (int i = 0; i < appFuncArray.length; i++)
{
    IApplicationFunction retAppFunc =
        (IApplicationFunction) appFuncArray[i];

    System.out.println ("appFuncArray[" + i +
        "].getName(): " +
        retAppFunc.getName());
}

// Get the ApplicationFunction we created by its name.
IApplicationFunction retByNameAppFunc =
    (IApplicationFunction) appFuncSparseData.
    getByName (appFunc.getName());
System.out.println ("retByNameAppFunc: " +
    retByNameAppFunc);

System.out.println ("Deleting ApplicationFunction: " +
    appFunc.getName());
appFunc.delete();
```

Example continues:

```

        System.out.println ("ApplicationFunction deleted");

        System.out.println ("Deleting Application:  " +
                            app.getName());
        app.delete();
        System.out.println ("Application deleted");

        // Disconnect from the APIServer.
        System.out.println ("Disconnecting from the APIServer");
        myServerProxy.disconnect();
        System.out.println ("Disconnected");
    }
    catch (Exception e)
    {
        e.printStackTrace();
    }
}

```

SmartRule Example

This Administrative API example, `SmartRuleExample.java`, creates a `SmartRule` and tests it on a user. The `SmartRule` will allow access to the "TestApp" application if the user's "employeeType" Property contains the word "Sales".

Before you begin, you must take the following steps:

- Create the following data items in your data store. In most cases, you will need to run the ClearTrust Entitlement Manager application and save each item.
 - a user called `TestUser`
 - a user property definition called `employeeType`
 - a Web server called `TestWebServer`
 - an application called `TestApp`
 - an `ApplicationURL`, `"/testpage.html"` on the `TestWebServer`
 - add the `ApplicationURL` `"/testpage.html"` to the `TestApp` application
- In your LDAP administration tool (for example, the iPlanet Console if you use iPlanet) make sure the "employeeType" attribute is available (writeable) on your user entries. Unless you have modified the default iPlanet schema, `employeeType` should be available. This is a standard attribute, so you should not have to define the attribute from scratch.

For instructions on compiling examples, see ["Compiling Applications"](#) on page 68.

SmartRule Example

```

package sirrus.samples.admin;

import java.util.Date;

import sirrus.api.client.*;
import sirrus.api.client.criteria.*;

/**
 * SmartRuleExample
 *
 * @version 4.7
 * @since February 10, 2001
 */
public class SmartRuleExample {
    public static void main (String[] args)
    {
        try
        {
            APIServerProxy myServerProxy =
                new APIServerProxy("localhost", // API Server name
                                   5601,        // API Server port
                                   false);      // Use SSL

            // Connect to the APIServerProxy.
            myServerProxy.connect("admin",
                                  "admin1234",
                                  "Default Administrative Group",
                                  "Default Administrative Role");
            System.out.println ("Connected");

            // Load the sample UserPropertyDefinition
            IUserPropertyDefinition userPropDef =
                (IUserPropertyDefinition)myServerProxy.
                getUserPropertyDefinitions().
                getByName("employeeType");
            System.out.println ("Got the UserPropertyDefinition");

            // Load the sample Application
            IApplication app = (IApplication)
                myServerProxy.getApplications().
                getByName("TestApp");
            System.out.println ("Got the Application");
            System.out.println (app.getName() +
                                " has resources " +
                                app.getApplicationURLs().toString());
        }
    }
}

```


Example continues:

```
// Create and save the SmartRule
StringCriterion sampleCriterion = new
    StringCriterion(StringCriterion.CONTAINS, "Sales");
ISmartRule smartRule = app.createSmartRule(userPropDef);
smartRule.setCategory(ISmartRule.ALLOW);
smartRule.setSmartRuleCriterion(sampleCriterion);
smartRule.save();
System.out.println ("Created and saved the SmartRule");

// Load the sample User
IUser user = (IUser)
    myServerProxy.getUsers().getByName("TestUser");
System.out.println ("Got the User");

// Set the User's Property and save
user.setUserProperty("employeeType", "Inside Sales");
user.save();
System.out.println ("Changed the User's UserProperty");

// Entitlement changes have occurred; must clear caches.
myServerProxy.flushCache();

// Test access
String userName = "TestUser";
String password = null; //Password not needed; leave null
String webServerName = "TestWebServer";
String uriString = "/testpage.html";

boolean hasAccess = false;
hasAccess = myServerProxy.checkAccess(userName,
                                     password,
                                     webServerName,
                                     uriString);

myServerProxy.disconnect();

if (hasAccess)
{
    System.out.println ("Success!\n" +
                       userName +
                       " has access privileges to " +
                       uriString + " on " + webServerName);
}
```

Example continues:

```

        else
        {
            System.out.println ("Failure!\n" +
                                userName +
                                " should have access privileges to " +
                                uriString + " on " + webServerName);
            System.out.println ("Possible causes are:\n" +
                                "1. The user or a parent group " +
                                "has an overriding " +
                                "Entitlement assigned to it.\n" +
                                "2. The user's account has expired.\n" +
                                "3. The user is locked out.");
            return;
        }
    }
    catch (Exception e)
    {
        e.printStackTrace();
    }
}

```

User Search Example

This Administrative API example, `UserSearchExample`, shows you how to use the `Search` and `Criteria` packages to look up users. These search utilities work in the same way for most other administrative objects such as `IUserProperties` and `IWebServers`. For more information, see the Javadocs for the `sirrus.api.client.search` package.

Before you begin, you may need to do some of the following:

- Make sure the following exist in your LDAP data store. In most cases, you will need to run the RSA ClearTrust Entitlements Manager application and save each item.
 - a `UserPropertyDefinition` called `employeeType`
 - a `UserPropertyDefinition` called `initials`
 - a user with values set for the `employeeType` and `initials` properties. You can edit the property values or edit the search strings in this program so that the search will succeed.
- In your LDAP administration tool (e.g. Netscape Console) make sure the `employeeType` and `initials` attributes are writable on your user entries.

For instructions on compiling this example, see [“Compiling Applications”](#) on page 68.

UserSearchExample

```

package sirrus.samples.admin;

import sirrus.api.client.*;
import sirrus.api.client.search.*;
import sirrus.api.client.criteria.*;

/**
 * UserSearchExample.java
 *
 * @version 4.7
 * @since October 23, 2001
 */
public class UserSearchExample {
    public static void main (String[] args)
    {
        try
        {
            APIServerProxy myServerProxy = new
                APIServerProxy("localhost",
                    5601,
                    false);

            // Connect to the APIServerProxy.
            myServerProxy.connect("admin",
                "admin1234",
                "Default Administrative Group",
                "Default Administrative Role");
            System.out.println ("Connected");

            // Obtain a UserSearch object.
            IUserSearch userSearch =
                myServerProxy.searchUserObjects();

            // Add a first UserPropertyCriterion, which will check
            // if the value of the 'employeeType' property starts with
            // the string "Outside".
            userSearch.putUserPropertyCriterion(new
                UserPropertyCriterion("employeeType", new
                    StringCriterion(StringCriterion.STARTS_WITH,
                        "Outside")));

            // Add another UserPropertyCriterion, which will check
            // if the value of the 'initials' property contains
            // the string "(TU)".
            userSearch.putUserPropertyCriterion(new
                UserPropertyCriterion("initials", new
                    StringCriterion(StringCriterion.CONTAINS,
                        "TU")));
        }
    }
}

```

Example continues:

```
// Obtain the SparseData of Users matching the
// specified Criteria.
ISparseData usersInSearch = userSearch.search();

// Find out how many users are in the SparseData
int numofUsers = 0;
try
{
    IAPIObject[] retObj = usersInSearch.getByRange(0, Integer.MAX_VALUE);
    numofUsers = retObj.length;
    System.out.println ("Number of Users returned "
        + "by the Search: "
        + (numofUsers));
} catch (BadArgumentException e) {
    e.printStackTrace();
} catch (TransportException e) {
    e.printStackTrace();
} catch (java.io.IOException e) {
    e.printStackTrace();
}

// Get the objects by range and print them.
try{
    IAPIObject[] userArray =
        usersInSearch.getByRange(0, numofUsers);
    for (int i = 0; i < userArray.length; i++)
    {
        IUser returnedUser = (IUser)userArray[i];
        if(returnedUser.getName() != null)
        {
            System.out.println ("Loaded User: " +
                returnedUser.getFirstName() +
                " " +
                returnedUser.getLastName());
        }
    }
} catch (BadArgumentException e) {
    e.printStackTrace();
} catch (TransportException e) {
    e.printStackTrace();
} catch (java.io.IOException e) {
    e.printStackTrace();
}
}
catch (Exception e)
{
    e.printStackTrace();
}
}
```


5

Runtime C API

The RSA ClearTrust® Runtime API allows trusted C clients to perform authentication, authorization, and other functions using the runtime functionality of the RSA ClearTrust Authorization Servers. Using the information in this chapter together with the function descriptions in the header files, developers can build custom authentication or personalization programs that use or extend the RSA ClearTrust system's runtime functionality.

This Chapter

This chapter consists of:

- The “[C Runtime API Overview](#)”, below.
- Compilation instructions starting with “[Installing and Compiling](#)” on page 134.
- Instructions for connecting a Runtime API client, in “[Connecting a Runtime C API Client](#)” on page 135.
- The “[Runtime C API Reference](#)” starting on page 141.
- An example program:
 - “[RSA SecurID Authentication Example](#)” on page 153

The source code for the API example program is installed in

```
<CT_HOME>/api/runtime-c/example
```

C Runtime API Overview

The runtime functionality of the RSA ClearTrust Authorization Servers consists of four main pieces:

- Authentication of users
- Authorization of users to resources
- SSO (single sign-on) token creation & manipulation
- User property retrieval

Runtime tasks are read-only; that is, they perform queries on the existing state of the data in the RSA ClearTrust Servers, but do not involve changing or adding data. For tasks involving updates, the RSA ClearTrust Administrative API should be used; see [Chapter 3, “Administrative C API”](#). Both APIs may be used in a single client program.



Note: This chapter does not provide comprehensive function descriptions. See the header files for function descriptions.

Authentication

The credentials required for a user authentication operation depend on the type of authentication. The Runtime API supports these authentication types:

- RSA ClearTrust basic authentication
- RSA SecurID authentication
- NT authentication.
- LDAP authentication (Note that in RSA ClearTrust 4.7, this authentication type is equivalent to RSA ClearTrust basic authentication, since basic authentication verifies the user's password against an LDAP datastore. In future releases, basic authentication will verify the password against whatever type of underlying datastore you have configured to store users.)

In addition to the authentication types listed above, the API provides a mechanism to extend the Authorization Servers to include additional custom forms of user authentication.



Note: Note that in version 4.7 the Runtime API *does not* support Certificate Authentication. Previous versions of the Runtime API did support Certificate Authentication (and in 4.7 the *Web Server Agents* still do support Certificate Authentication; only the *API* does not).

Authorization

Authorization checks may be performed for either a Web resource (URL) or, more generically, for a ClearTrust application function. Such checks invoke the full RSA ClearTrust access control policy model, including both basic entitlements and SmartRules.

The implementation of this API is based on network communications with one or more RSA ClearTrust Authorization Servers. Clients may choose among a variety of connection options, from using a single Authorization Server connection to building a pool of connections to multiple Authorization Servers. Connection pools automatically and transparently fail over requests in the event of a server crash or network outage, and can be configured to distribute requests in a round-robin fashion. Depending on system configuration, the Runtime API's connections to the Authorization Server may be protected using Anonymous SSL or Authenticated SSL.

SSO Token Manipulation

To communicate authentication data between components, the Runtime API creates or updates a user's SSO (Single Sign-on) token when that user successfully authenticates. These tokens are passed between the Authorization Server, Web servers, application servers, and custom applications as a means of providing single sign-on functionality.

Since SSO tokens contain sensitive information, the system may be set to require that Runtime API clients connect over SSL before they can access token data. See [“SSL and Non-SSL Connection Options”](#) on page 136.

Runtime API clients may also instruct the server to implicitly return the user's token each time a user is authenticated or authorized. To do this, the client must set the `SC_TOKENS_ENABLED` flag to *true* when creating the Authorization Server connection. You can set this flag in the `credentials` parameter of the `ct_create_pool_with_dispatchers_ext` function. (See `ct_runtime_api.h` for more details.)

User Property Retrieval

In addition to checking and communicating authentication and authorization for users, the Runtime API provides access to users' exportable custom properties. These are the same user properties that are used in SmartRules. Because some properties may be deemed too sensitive to externalize, only user properties which are defined to be exportable are visible to Runtime API clients. (The “exportable” concept was introduced in version 4.6.)

Since user properties can contain sensitive information, the system may be set to require that Runtime API clients connect over SSL before they can access properties. See [“SSL and Non-SSL Connection Options”](#) on page 136.

Runtime API clients may also instruct the server to implicitly return the user's properties each time a user is authenticated or authorized. To do this, the client must set the `SC_USER_PROPERTIES_ENABLED` flag to *true* when creating the Authorization Server connection. You can set this flag in the `credentials` parameter of the `ct_create_pool_with_dispatchers_ext` function. (See `ct_runtime_api.h` for more details.) Because user property retrieval slows the execution of an authentication or authorization call, this flag defaults to *false*.

For information on creating user properties in your LDAP user data store, see the “Data Integrity” section in the “Installing the LDAP Data Adapters” chapter of the *Installation and Configuration Guide*.

Installing and Compiling

This section explains the installed components that make up the API and provides guidelines for building applications. For instructions on installing the APIs, see [Chapter 2, “Installing the RSA ClearTrust APIs”](#).

Location

The files you need to build and run Runtime API applications are installed when the RSA ClearTrust Servers are installed. The files you need are in the following directories:

```
<CT_HOME>/api/runtime-c/include
                        /lib
                        /example
```

- `include` contains the API header files. See the next section for descriptions.
- `lib` contains the API libraries
- `example` contains the example code from this chapter.

Header Files

API Files

The following header files define the methods and types of the RSA ClearTrust Runtime API.

- `ct_runtime_api.h` this is the main file of the Runtime API. It contains the `ct_authenticate()` and `ct_authorize()` functions, functions to establish a Runtime API client connection, and functions to work with user SSO tokens and user properties.
- `ct_auth_result.h` defines structs used to return authentication and authorization results as well as the `ct_auth_result_tostring()` method for converting `ct_auth_result` structures to strings.
- `ct_auth_types.h` establishes and documents the types of authentication supported in the RSA ClearTrust system.
- `ct_boolean.h` defines the ClearTrust boolean type for use on unix and Windows.
- `ct_callback.h` contains the callback function for obtaining the passphrase for unlocking the keystore when SSL is used between the C Runtime API client and the RSA ClearTrust servers.
- `ct_error.h` defines structs used to return error messages as well as the `ct_error_tostring()` method for converting `ct_error` structures to strings.

- `ct_hash.h`, `ct_lock.h`, `ct_lock_impl.h`, and `ct_windows.h` are included for compilation only and do not include any exposed functions.
- `ct_map.h` and `ct_map_utils.h` contain utilities for working with Maps. Maps are multivalued datatypes used in the RSA ClearTrust system to describe objects like users and authentication results.
- `ct_pool_manager.h` contains methods for creating Authorization Server pools that allow Runtime API clients to connect to RSA ClearTrust Authorization Servers in a scalable manner that provides some failover.
- `ct_resource_constants.h` provides keys to indicate the type of an RSA ClearTrust-protected resource.
- `ct_result_constants.h` establishes and documents the return values that are used to indicate the success or failure of an authentication and/or authorization attempt in the RSA ClearTrust system.
- `ct_return_code_keys.h` established the types of return values that the Authorization Server can produce.
- `ct_token_keys.h` establishes the fields of the RSA ClearTrust SSO token.
- `ct_user_constants.h` contains the constant strings to be used as keys or values in `ct_map` objects representing RSA ClearTrust users and their authentication credentials in calls to the Runtime API.

Connecting a Runtime C API Client

Calls in the Runtime API are based on connection pools. A connection pool, also known as an Authorization Server pool, is a grouping of Authorization Servers. The pool is a mechanism for getting a connection to an Authorization Server. For most functions that use an Authorization Server connection, you will find an equivalent function that takes a pool as an argument, rather than a reference to a specific Authorization Server.

For routines that require an Authorization Server, using an Authorization Server pool is more fail-safe than naming a specific Authorization Server, since that named Authorization Server might be down. If an Authorization Server goes down, the pool will look for another based on a list of Authorization Servers provided by a Server Dispatcher. For details, see the descriptions of the `ct_create_pool_with_dispatcher` (see page 138) and `ct_create_pool_with_dispatcher_ext` (see page 138) functions. (See also `ct_runtime_api.h`.)

Before making any of the Runtime API calls that require a pool, the user needs to create a connection pool using the pool manager (see also `ct_pool_manager.h`). The pool manager is the collection of methods used to create and use Authorization Server pools. The pool manager is based on a simple hash table, with an integer as its key. This allows multiple Authorization Server pools to be used. The client only gets a hash code to the pool. Access to the pools is allowed only through these codes.

SSL and Non-SSL Connection Options

An RSA ClearTrust C Runtime API client may connect as an anonymous SSL client or as a clear text (non-SSL) client. Anonymous SSL clients are not authenticated but communicate over an encrypted SSL connection. Non-SSL clients communicate without encryption, in clear text.



Note: In order to assure the correct functioning of C Runtime API clients, make sure that the `cleartrust.dispatcher.list_port.force_anonymous` parameter is set to true in the Dispatcher's configuration file. This parameter is intended mainly to support future functionality in the Runtime API, but must be set to true in 4.7.

This section explains the cases in which you may wish to require SSL connections and how to connect using the three approaches.

Access to Tokens and User Properties

In order to protect the sometimes sensitive information in SSO tokens and user properties, RSA ClearTrust can be configured to check that a Runtime API client has established a secure enough connection before returning a token or property value.

This configuration is done by setting the `cleartrust.runtime_api.security` parameter in your Authorization Server's configuration file (`aserver.conf`). This parameter sets the minimum security required on the connection between the Runtime API client and the Authorization Server in order to create/manipulate SSO tokens and to retrieve user properties. Valid settings, in order of increasing security, are:

- `cleartext` for a minimum connection type of *clear text*. This allows tokens/properties to be passed over any type of connection.
- `anonymous` for a minimum connection type of *anonymous SSL*. This allows tokens/properties to be passed over anonymous or authenticated SSL connections.
- `authenticated` for a minimum connection type of *authenticated SSL*. This is the default. This allows tokens/properties to be passed over authenticated SSL connections only.

The following table summarizes the available levels of required security for connections. In this table, a "Yes" indicates that the Runtime API client of that row will be permitted to view tokens/user properties by an Authorization Server

configured with the `cleartrust.runtime_api.security` setting noted in that column.

Table 5.1 Access matrix for Runtime API clients wishing to access tokens/user properties

			Level of security required by Authorization Server (per setting of <code>cleartrust.runtime_api.security</code>)		
			cleartext	anonymous	authenticated
Runtime API client's connection type	cleartext	<code>cleartrust.net.ssl.use=NO</code> <code>cleartrust.net.ssl.require_authentication=false</code>	Yes	No	No
	anonymous	<code>cleartrust.net.ssl.use=YES</code> <code>cleartrust.net.ssl.require_authentication=false</code>	Yes	Yes	No
	authenticated	<code>cleartrust.net.ssl.use=YES</code> <code>cleartrust.net.ssl.require_authentication=true</code>	Yes	Yes	Yes

Token Access Example: No SSL

If the Authorization Server is configured with no SSL (in the `aserver.conf`, set `cleartrust.net.use.ssl=No`) and you set

```
cleartrust.runtime_api.security=anonymous
```

or

```
cleartrust.runtime_api.security=authenticated
```

then all Runtime API clients connecting to this Authorization Server will *not* be able to create/manipulate tokens or retrieve user properties. Only with this parameter set to `cleartext` will your system permit Runtime API clients to access tokens and user properties.

Token Access Example: Authenticated SSL

If the Authorization Server is configured to use authenticated SSL (the most secure type of connection), then, regardless of how you set `cleartrust.runtime_api.security`, the Runtime API clients *will* be able to create/manipulate SSO tokens and retrieve user properties.

Connection Pool Functions and Keys

Two tables follow. The first explains the functions related to connection pools, and the second explains the key settings that you can use to configure your connections.

Table 5.2 Connection Pool Functions

Function	Description
ct_create_pool_with_dispatcher	Creates a connection pool, using the ClearTrust Dispatcher for the list of servers. This is typically the way to create a pool. If this is used, depending on whether the pool is in round robin mode, the pool takes care of finding a valid connection to an Authorization Server. For user property retrieval and SSO token creation and manipulation, use instead ct_create_pool_with_dispatchers_ext, described below.
ct_create_pool_with_dispatchers_ext	Creates an Authorization Server connection pool using one Server Dispatcher from its list of Dispatchers to provide the current list of active Authorization Servers. The pool created with this method will poll the active Server Dispatcher periodically to get the latest list of active Authorization Servers. If that Dispatcher goes down, the pool will fail over to the next specified Dispatcher in its list. See the sd_host_names and sd_ports parameters. The pool can be set to provide Authorization Servers based on a geographic (or other) preference list. See the preference_list parameter. Pools created with this method can use authenticated SSL or clear text for communication. See the use_ssl parameter. If you wish to create a pool that uses anonymous SSL, you must use the ct_create_pool_with_dispatcher function instead. If you wish to work with users' SSO tokens and user properties, you must connect using authenticated SSL (use_ssl = true). In addition, if needed, you can turn on implicit token and/or user property retrieval, which forces every authentication or authorization call to return the user's SSO token and/or user properties. For details, see CT_TOKENS_ENABLED and CT_USER_PROPERTIES_ENABLED in ct_pool_manager.h, as well as “SSO Token Manipulation” and “User Property Retrieval” on page 133.
ct_create_pool_with_dispatchers_ext_v2	Works like ct_create_pool_with_dispatchers_ext, but also allows the user to provide parameters for a routine for acquiring passphrases to unlock keystore, and for opaque data returned to callback. This additional information is passed in the “opaque” field.
ct_create_blank_pool	Creates a connection pool, without a server_dispatcher address or any servers. Creating a pool in this way allows the user to add servers from their code, instead of relying on a ClearTrust Dispatcher to provide a list.
ct_create_blank_pool_ext	Creates a connection pool, without a server_dispatcher address or any servers. This function differs from the ct_create_blank_pool function in that pools created with this function can use a geographic (or other) preference_list to specify that some Authorization Servers on the list are preferable to others based on location or some other criterion.

Table 5.2 Connection Pool Functions

Function	Description
ct_add_server_to_pool	Used with ct_create_blank_pool to add a server to the given pool. Note that one of the arguments passed to this function is an output argument, ct_server_id . When calling this method, you must allocate a char* buffer and pass it as the ct_server_id argument. This buffer will hold the new id describing the Authorization Server. This is the key in the pool's server table. RSA recommends a declaring a buffer of 1024 bytes for this. See the header file, ct_pool_manager.h, for details.
ct_add_server_to_pool_ext	Adds a new auth_server, which could have a location class, to the given server pool. See note about ct_server_id in the description of ct_add_server_to_pool, above.
ct_add_server_to_pool_ext_v2	Adds a new auth_server, which could have a location class, to the given server pool. This version supports the opaque data field used to pass the keystore passphrase. See note about ct_server_id in the description of ct_add_server_to_pool, above.
ct_init_pool_table	Initializes the pool.
ct_lookup_pool	Attempts to locate the given pool index in the pool table.
ct_refresh_pool	Refresh the given connection pool by getting a new server list from the server list provider and creating new Authorization Server connection. Pending API requests are not affected.
ct_shutdown_pool	Closes all open connections in a given connection pool. Does not destroy the pool, just closes all connections.
ct_destroy_pool	Closes all open connections in a specified connection pool and frees all associated memory.
ct_print_pool	Prints the contents of the pool to stderr.
ct_remove_server_from_pool	Removes an Authorization Server connection from a specified pool.
ct_release_server_reference	In a multithreaded scenario, a connection to a particular Authorization Server could close if another thread refreshes the connection pool. To avoid this scenario, anytime a client is using a connection, it should have a server reference. Once the client is done with the server, this call should be made to 'release' the reference, and allow any waiting threads to refresh the pool.
ct_grab_server_reference	Gets a key to a server in the specified pool.

Table 5.3 Connection Pool Keys

Key	Description
CT_SSL_KEYSTORE	This key contains the name of the PKCS12 keystore containing certificates/private key.
CT_SSL_CALLBACK	Used for the callback function that retrieves the passphrase to unlock keystore. See <code>ct_callback.h</code> for details.
CT_TOKENS_ENABLED	This key in the credentials map enables/disables the implicit token manipulation functionality of the Runtime API for the current connection. Set this to true to enable the token functionality; false otherwise. When set to true, authentication calls will return tokens. If this field is not set, the behavior defaults to false.
CT_USER_PROPERTIES_ENABLED	This key in the credentials map enables/disables the user property retrieval functionality of the Runtime API for the current connection. Set this to true to enable the functionality; false otherwise. When set to true, authentication calls will return the user's exportable properties. If this field is not set, the behavior defaults to false.
CT_CALLBACK_KEYSTORE_PASSPHRASE	Reason code for SSL/keystore callback routine. Indicates a request for the passphrase used to unlock the keystore.
CT_CALLBACK_PRIVATE_KEY_PASSPHRASE	Reason code for SSL/keystore callback routine. Indicates a request for the passphrase used to unlock the private key.

Runtime C API Reference

Client Keys

The constant strings used for describing users and credentials are defined in the `ct_user_constants.h` header. These strings may be used as keys or values in `ct_map` objects representing ClearTrust users and their authentication credentials in calls to the Runtime API.

All user maps must contain a value for `CT_RUNAPI_USER_ID_KEY`, `CT_RUNAPI_USER_DN_KEY`, `CT_RUNAPI_USER_CERT_KEY`, or `CT_RUNAPI_TOKEN_KEY`, but not more than one of these. Whichever value is present will determine how the user is identified in ClearTrust:

- by user ID,
- by distinguished name,
- by the DN contained in a certificate, or
- by the ID contained in a SSO token, respectively.

In addition, for calls to `ct_authenticate()` or `ct_authenticate_pool()`, a value for `CT_RUNAPI_AUTHENTICATION_TYPE` must be present for the user map to be valid. This key-value pair may also be optionally provided in calls to `ct_authorize()` or `ct_authorize_pool()`.

Table 5.4 Client Keys

Key	Description
<code>CT_RUNAPI_USER_ID_KEY</code>	The key for the ID of a user, as known to RSA ClearTrust. A valid user map must have a value of non-zero length for this key, for <code>CT_RUNAPI_USER_DN_KEY</code> , for <code>CT_RUNAPI_USER_CERT_KEY</code> , or for <code>CT_RUNAPI_TOKEN_KEY</code> , but ONLY one of these. Whichever value is present will determine how the user is identified in RSA ClearTrust.
<code>CT_RUNAPI_USER_DN_KEY</code>	The key for the distinguished name of a user, as known to RSA ClearTrust. A valid user map must have a value of non-zero length for this key, for <code>CT_RUNAPI_USER_ID_KEY</code> , for <code>CT_RUNAPI_USER_CERT_KEY</code> , or for <code>CT_RUNAPI_TOKEN_KEY</code> , but ONLY one of these. Whichever value is present will determine how the user is identified in RSA ClearTrust.
<code>CT_RUNAPI_USER_CERT_KEY</code>	The key for the cert of a user, as known to RSA ClearTrust. A valid user map must have a value of non-zero length for this key, for <code>CT_RUNAPI_USER_ID_KEY</code> , for <code>CT_RUNAPI_USER_DN_KEY</code> , or for <code>CT_RUNAPI_TOKEN_KEY</code> , but ONLY one of these. Whichever value is present will determine how the user is identified in RSA ClearTrust.

Table 5.4 Client Keys

Key	Description
CT_RUNAPI_TOKEN_KEY	<p>The key for this user's authentication token. A valid user map must have a value of non-zero length for this key, for CT_RUNAPI_USER_ID_KEY, for CT_RUNAPI_USER_DN_KEY, or for CT_RUNAPI_USER_CERT_KEY, but ONLY one of these. Whichever value is present will determine how the user is identified in RSA ClearTrust. In order for a user to be identified by a token, the token functionality must be enabled (see discussion of mutually-authenticated SSL in ct_runtime_api.h). If a token is used as the user ID in an operation involving authentication (i.e. authenticate() or authorize()) with CT_RUNAPI_AUTHENTICATION_TYPE specified) and the authentication succeeds, the token returned under CT_AUTH_TOKEN_STR will be an updated version of this token.</p>
CT_RUNAPI_AUTHENTICATION_TYPE	<p>The value of this key indicates what type of authentication is being requested in a call to ct_authenticate() or authorize().</p> <p>Currently, the permissible built-in values for CT_RUNAPI_AUTHENTICATION_TYPE are SC_AUTH_TYPE_BASIC, SC_AUTH_TYPE_NT, SC_AUTH_TYPE_LDAP, SC_AUTH_TYPE_CERT, and SC_AUTH_TYPE_SECURID (all defined in ct_auth_types.h), and SC_AUTH_TYPE_USER_CHECK (defined below). This last value is used merely to verify the existence of a user in the Entitlements database. No credentials need be supplied. The other forms of authentication each require some credentials to succeed. The appropriate credentials should be packed into a string and placed into the user map under the key CT_RUNAPI_CREDENTIALS_KEY.</p> <p>Client programs may alternatively supply a custom value. In this case, the auth server will look for and drive the class <code>sirrus.runtime.customauth.Authenticator</code> whose <code>getAuthenticationType()</code> method returns the same custom value. The Authenticator can then make use of whatever custom credential values are appropriate to that authentication type. To avoid conflicts with pre-defined auth types, custom values should not begin with the prefix "SC_".</p> <p>A value for this key must be present in the user map passed to <code>ct_authenticate()</code> or <code>ct_authenticate_pool()</code>, but is optional for <code>ct_authorize()</code> or <code>ct_authorize_pool()</code>.</p> <p>In the latter case, if it is not present the authentication type implicitly defaults to SC_AUTH_TYPE_USER_CHECK. In other methods that require a user map argument (such as <code>ct_create_token()</code> and <code>ct_create_token_pool()</code>), any value supplied for this key will be ignored.</p>

Table 5.4 Client Keys

Key	Description
SC_AUTH_TYPE_USER_CHECK	<p>The value to use for CT_RUNAPI_AUTHENTICATION_TYPE to drive a simple user validation check. This built-in authentication method validates that the user exists in the RSA ClearTrust database. No additional credential need be supplied. Note that this check is also implicitly made for each of the built-in authentication types, but is NOT made for any custom authentication type.</p> <p>The possible return codes for this type of authentication are: VALID_USER UNKNOWN_USER INACTIVE_ACCOUNT EXPIRED_ACCOUNT ADMIN_LOCKOUT</p>
CT_RUNAPI_CREDENTIALS_KEY	<p>The key for a user's credentials. All of the currently supported built-in authentication types except for SC_AUTH_TYPE_USER_CHECK (i.e. SC_AUTH_TYPE_BASIC, SC_AUTH_TYPE_LDAP, SC_AUTH_TYPE_NT, SC_AUTH_TYPE_CERT and SC_AUTH_TYPE_SECURID) require an appropriate value for this key. (Another exceptional case is when CT_RUNAPI_SECURID_NEW_PIN_KEY or CT_RUNAPI_SECURID_NEXT_CODE_KEY is used. Read their descriptions below for more details.)</p> <p>Implementations of custom authentication types may re-use this key or define their own.</p>
CT_RUNAPI_SECURID_NEW_PIN_KEY	<p>The key for a user's new SecurID pin. This is used by the SC_AUTH_TYPE_SECURID authentication type. The CT_RUNAPI_CREDENTIALS_KEY should not be set in this situation. This field is ONLY used when the token is in the CT_AUTH_NEW_PIN_REQUIRED_STR mode and the user needs to provide his new pin to the system. This field contains the user's desired new pin only, not the user's whole SecurID passcode.</p>

Table 5.4 Client Keys

Key	Description
CT_RUNAPI_SECURID_NEXT_CODE_KEY	<p>The key for a user's SecurID next code. This field should contain the SecurID tokencode only, which does not include user's pin. This is used by the SC_AUTH_TYPE_SECURID authentication types. The CT_RUNAPI_CREDENTIALS_KEY should not be set in this situation. This field is ONLY used when the token is in the CT_AUTH_NEXT_CODE_REQUIRED_STR mode and the user needs provide his next SecurID tokencode to the system.</p>
CT_RUNAPI_SERVER_STATE	<p>The key indicating the user's current authentication state in a multistate authentication routine.</p> <p>When doing SecurID authentication, if an initial authentication attempt results in the SecurID ACE Agent requesting the next code or a new pin, the Authorization Server will return a state token to the api user, with the key CT_AUTH_SERVER_STATE set. When the Runtime API client makes its follow-on authentication attempt, it MUST insert the CT_AUTH_SERVER_STATE into this field, CT_RUNAPI_SERVER_STATE, and submit this state along with the next code or new pin information.</p> <p>NOTE that the ct_create_user_securid_*_map functions will take care of the details of doing the map.</p> <p>The CT_RUNAPI_SERVER_STATE field should be thought of as opaque to the Runtime API client. The actual contents of the field are a set of values that guarantee the correct Authorization Server will service the request.</p> <p>See also CT_AUTH_SERVER_STATE in ct_result_constants.h</p>

Authentication Types

The authentication types are defined in `ct_auth_types.h`. Their descriptions are reprinted in the sections that follow.

SC_AUTH_TYPE_BASIC

This constant denotes the RSA ClearTrust basic authentication mode. The submitted user name and password are verified against those stored in the RSA ClearTrust user datastore.

For this type of authentication, the other needed properties are (from `ct_user_constants.h`):

- `CT_RUNAPI_USER_ID_KEY` (or other user identifier)
- `CT_RUNAPI_CREDENTIALS_KEY`

For this type of authentication, the possible values for the key `CT_AUTH_AUTHENTICATION_RESULT_STR` in maps returned from `ct_authenticate()`, `ct_authenticate_pool()`, `ct_authorize()`, or `ct_authorize_pool()` are (from `ct_result_constants.h`):

- `CT_AUTH_VALID_USER_STR`
- `CT_AUTH_UNKNOWN_USER_STR`
- `CT_AUTH_INACTIVE_ACCOUNT_STR`
- `CT_AUTH_EXPIRED_ACCOUNT_STR`
- `CT_AUTH_ADMIN_LOCKOUT_STR`
- `CT_AUTH_INVALID_PASSWORD_STR`
- `CT_AUTH_EXPIRED_PASSWORD_STR`
- `CT_AUTH_EXPIRED_PASSWORD_NEW_USER_STR`
- `CT_AUTH_EXPIRED_PASSWORD_FORCED_STR`

SC_AUTH_TYPE_NT

This constant denotes the Windows NT authentication mode. The submitted user name and password are verified against those stored in the Microsoft Windows PDC/BDC user account records. For this type of authentication, the other needed properties are (from `ct_user_constants.h`):

- `CT_RUNAPI_USER_ID_KEY` (or other user identifier)
- `CT_RUNAPI_CREDENTIALS_KEY`
- `CT_RUNAPI_NT_DOMAIN_KEY`

For this type of authentication, the possible values for the key `CT_AUTH_AUTHENTICATION_RESULT_STR` in maps returned from `ct_authenticate()`, `ct_authenticate_pool()`, `ct_authorize()`, or `ct_authorize_pool()` are (from `ct_result_constants.h`):

- CT_AUTH_VALID_USER_STR
- CT_AUTH_INACTIVE_ACCOUNT_STR
- CT_AUTH_EXPIRED_ACCOUNT_STR
- CT_AUTH_ADMIN_LOCKOUT_STR
- CT_AUTH_NT_AUTH_FAILED_STR
- CT_AUTH_EXPIRED_PASSWORD_STR

SC_AUTH_TYPE_CERT (not supported in RSA ClearTrust 4.7)



Note: Note that in version 4.7 the Runtime API *does not* support Certificate Authentication. Previous versions of the Runtime API did support Certificate Authentication (and in 4.7 the *Web Server Agents* still do support Certificate Authentication; only the *API* does not).

The SC_AUTH_TYPE_CERT constant denotes the Certificate authentication mode. For this type of authentication, the other needed properties are (from ct_user_constants.h):

CT_RUNAPI_USER_CERT_KEY

For this type of authentication, the possible values for the key CT_AUTH_AUTHENTICATION_RESULT_STR in maps returned from ct_authenticate(), ct_authenticate_pool(), ct_authorize(), or ct_authorize_pool() are (from ct_result_constants.h):

- CT_AUTH_VALID_USER_STR
- CT_AUTH_UNKNOWN_USER_STR
- CT_AUTH_INACTIVE_ACCOUNT_STR
- CT_AUTH_EXPIRED_ACCOUNT_STR
- CT_AUTH_ADMIN_LOCKOUT_STR

SC_AUTH_TYPE_SECURID

This constant denotes the SecurID authentication mode. For this type of authentication, the other needed properties that must be passed as arguments are:

(a) For the first try:

- CT_RUNAPI_USER_ID_KEY (or other user identifier)
- CT_RUNAPI_CREDENTIALS_KEY containing SecurID passcode

(b) When the returned value of CT_AUTH_AUTHENTICATION_RESULT_STR is CT_AUTH_NEW_PIN_REQUIRED_STR:

- CT_RUNAPI_USER_ID_KEY (or other user identifier)
- CT_RUNAPI_SECURID_NEW_PIN_KEY

- CT_AUTH_SERVER_STATE

(c) When the returned value of CT_AUTH_AUTHENTICATION_RESULT_STR is CT_AUTH_NEW_PIN_ACCEPTED_STR:

- CT_RUNAPI_USER_ID_KEY (or other user identifier)
- CT_RUNAPI_CREDENTIALS_KEY containing SecurID passcode
- CT_AUTH_SERVER_STATE

(d) When the returned CT_AUTH_AUTHENTICATION_RESULT_STR is CT_AUTH_NEXT_CODE_REQUIRED_STR:

- CT_RUNAPI_USER_ID_KEY (or other user identifier)
- CT_RUNAPI_SECURID_NEXT_CODE_KEY
- CT_AUTH_SERVER_STATE

For this type of authentication, the possible values for the key CT_AUTH_AUTHENTICATION_RESULT_STR in maps returned from ct_authenticate(), ct_authenticate_pool(), ct_authorize(), or ct_authorize_pool() are (from ct_result_constants.h):

- CT_AUTH_VALID_USER_STR
- CT_AUTH_INACTIVE_ACCOUNT_STR
- CT_AUTH_EXPIRED_ACCOUNT_STR
- CT_AUTH_ADMIN_LOCKOUT_STR
- CT_AUTH_NEXT_CODE_REQUIRED_STR
- CT_AUTH_NEW_PIN_REQUIRED_STR
- CT_AUTH_NEW_PIN_REJECTED_STR
- CT_AUTH_NEW_PIN_ACCEPTED_STR
- CT_AUTH_SECURID_AUTH_FAILED_STR

SC_AUTH_TYPE_LDAP

This constant denotes the LDAP authentication mode. This built-in authentication method validates that the user exists both in the ClearTrust database and in LDAP, and that the value supplied for CT_RUNAPI_CREDENTIALS_KEY is the correct LDAP password.

The user's identifier (i.e. whichever of CT_RUNAPI_USER_ID_KEY, CT_RUNAPI_USER_DN_KEY, CT_RUNAPI_TOKEN_KEY, or CT_RUNAPI_USER_CERT_KEY is present) must match the appropriate field in LDAP, as determined by the parameter cleartrust.aserver.ldapauth.ldapattr_map_scuid in the server-side webagent.conf file.

For this type of authentication, the other needed properties are (from ct_user_constants.h):

- CT_RUNAPI_USER_ID_KEY (or other user identifier)

- CT_RUNAPI_CREDENTIALS_KEY

For this type of authentication, the possible values for the key CT_AUTH_AUTHENTICATION_RESULT_STR in maps returned from ct_authenticate(), ct_authenticate_pool(), ct_authorize(), or ct_authorize_pool() are (from ct_result_constants.h):

- CT_AUTH_VALID_USER_STR
- CT_AUTH_UNKNOWN_USER_STR
- CT_AUTH_INACTIVE_ACCOUNT_STR
- CT_AUTH_EXPIRED_ACCOUNT_STR
- CT_AUTH_ADMIN_LOCKOUT_STR
- CT_AUTH_INVALID_PASSWORD_STR

SC_AUTH_TYPE_CUSTOM

SC_AUTH_TYPE_CUSTOM signifies a custom authentication type implemented as a WAX (Web Agent eXtension) authentication handler. See [“Invoking a WAX Authentication Method”](#) on page 215.

This constant may be used only as a key in SSO tokens, where it signifies that the user has already authenticated with a “CUSTOM” authentication method implemented in your Web Server Agent.

This constant cannot be used as a CT_RUNAPI_AUTHENTICATION_TYPE in user maps when requesting an authentication via the Runtime API, because this type of authentication is performed in the RSA ClearTrust Web Server Agent, not in the Authorization Server. The Runtime API provides access only to those authentications that are performed by the Authorization Server.

NOTE: SC_AUTH_TYPE_CUSTOM is NOT the same as custom authentication types implemented in the auth servers using the `sirrus.runtime.customauth` package.

Runtime Functions

The C Runtime API functions of `ct_runtime_api.h` provide access to all of the Authorization Server functionality. These functions fall into five major sections:

- authentication
- authorization
- token manipulation
- user property retrieval
- miscellaneous functions such as cache clearing.

The runtime functions are provided to work either directly with a specified Authorization Server, or to work with a server pool provided by a Dispatcher Server. The difference between them is that for a server pool, the pool is searched and an available Authorization Server supplied automatically.

Table 5.5 Runtime Functions: Authorization and Authentication

Function	Description
<code>init_ct_runtime_api_with_ssl</code>	<p>Initializes the Runtime API. You must call this method (or <code>init_ct_runtime_api</code>) before calling any other method in the Runtime API.</p> <p>This method takes a boolean input parameter, "initialize_ssl". If TRUE, this method will initialize the OpenSSL library as well as performing all other initializations required for the Runtime API. If FALSE, this method will initialize everything except OpenSSL.</p> <p>Skipping OpenSSL initialization is useful for runtime API clients running in environments in which SSL has already been initialized externally to the API. In particular, you must set this to FALSE for Runtime API clients running within the ClearTrust Web Server Agent Extension (WAX) environment.</p> <p>This method returns a boolean, TRUE if the initialization completed successfully, FALSE if the API has previously been initialized.</p>
<code>ct_authorize</code>	<p>Checks whether the specified user has permission to access the specified resource. The check is performed by the specified Authorization Server.</p> <p>When calling <code>ct_authorize()</code>, you must specify the user in the input <code>ct_map</code>, "user"; the requested resource in the input <code>ct_map</code>, "resource"; and the Authorization Server in the input <code>ct_map</code>, "server_key". The method will return the authorization results in an output <code>ct_map</code> called "auth_results", under the key "CT_AUTH_AUTHORIZATION_RESULT_STR."</p> <p>This and all authorization methods can be set to return the user's SSO token; see "SSO Token Manipulation" on page 133. Also, all authorization methods can be set to return the user's properties; see "User Property Retrieval" on page 133.</p>
<code>ct_authorize_pool</code>	<p>Checks whether the specified user has permission to access the specified resource. The check is performed by an Authorization Server from the specified server pool. Inputs and outputs are the same as those for <code>ct_authorize</code>, with the exception that, in this function, a <code>pool_key</code> is passed to specify an Authorization Server Pool, rather than a <code>server_key</code> indicating a specific Authorization Server.</p>

Table 5.5 Runtime Functions: Authorization and Authentication

Function	Description
ct_check_resource_status	Determines whether the specified resource is protected for the specified server.
ct_check_resource_status_pool	Determines whether the specified resource is protected for the specified server pool.
ct_authenticate	Determines whether the specified user is a valid user in the database using the specified server. This and all authentication methods can be set to return the user's SSO token; see “SSO Token Manipulation” on page 133. Also, all authentication methods can be set to return the user's properties; see “User Property Retrieval” on page 133.
ct_authenticate_pool	Determines whether the specified user is a valid user in the database using a server pool.
ct_clear_server_caches	Clears server caches for the specified server.
ct_clear_server_caches_pool	Clears server caches for a server in the server pool.
ct_map_get_default_auth_result	Returns an auth_result struct containing an error code, with a corresponding string, given a typical map of results from the Authorization Server (from a built-in call to the server).

Table 5.6 Runtime Functions: User Properties

Function	Description
ct_get_user_properties	Uses the specified server, and requests all exportable properties for the specified user.
ct_get_user_properties_pool	Uses the specified pool, finds an available server, and requests all exportable properties for the specified user.
ct_get_user_property	Uses the specified server, and requests a specific property by name for the specified user.
ct_get_user_property_pool	Uses the specified pool, finds an available server, and requests a specific property by name for the specified user.

Table 5.7 Runtime Functions: SSO Tokens

Function	Description
ct_create_token	Uses the specified server to create a new SSO token.
ct_create_token_pool	Uses the specified server pool to create a new SSO token.
ct_set_token_value	Uses the specified server to set a key-value pair in a token.
ct_set_token_value_pool	Uses the specified server pool to set a key-value pair in a token.
ct_get_token_value	Uses the specified server to get the value corresponding to a given key in a token.
ct_get_token_value_pool	Uses the specified server pool to get the value corresponding to a given key in a token.

Table 5.7 Runtime Functions: SSO Tokens

Function	Description
ct_set_token_values	Uses the specified server to set one or more key-value pairs in a token.
ct_set_token_values_pool	Uses the specified server pool to set one or more key-value pairs in a token.
ct_get_token_values	Uses the specified server to get all of the values contained in a token.
ct_get_token_values_pool	Uses the specified server pool to get all of the values contained in a token.
ct_validate_token	Uses the specified server to validate a token. A token is considered valid if it can be decrypted with one of the crypto keys currently active in the system. For a given valid token, this function returns a new token which is the same as the original but with the touch time updated.
ct_validate_token_pool	Uses the specified server pool to validate a token. A token is considered valid if it can be decrypted with one of the crypto keys currently active in the system. For a given valid token, this function returns a new token which is the same as the original but with the touch time updated.

Table 5.8 Runtime Functions: Utilities

Function	Description
ct_get_version	Returns a string indicating what version of RSA ClearTrust this is. This returns the version of the client-side code only (i.e. the server is not queried for its version).
ct_test_server	Checks whether the Authorization Server is still responding. Returns CT_OK if all is well.
ct_test_server_pool	Checks whether <i>all</i> the Authorization Servers in the pool are responding. Returns CT_OK only if <i>all</i> servers in the pool are alive.

Maps

Maps consist of key-value pairs. Maps are used as arguments to the Runtime API. For custom functionality, the maps themselves are provided to the client. The functions listed below are documented fully in the `ct_map.h` and `ct_map_utils.h` headers.

Table 5.9 Map Functions defined in `ct_map.h`

Function	Description
ct_create_map	Creates a map object.
ct_map_find	Gets the value associated with the specified map key.
ct_map_insert	Inserts a key-value pair into the specified map.
ct_map_remove	Removes a key-value pair from the specified map.
ct_map_destroy	Destroys the specified map, including all of its key-value pairs. See also <code>ct_destroy_three_maps</code> and <code>ct_destroy_two_maps</code> in <code>ct_map_utils.h</code> below.
ct_map_clear	Destroys all key-value pairs in the specified map, but leaves the map intact. See also <code>ct_clear_three_maps</code> in <code>ct_map_utils.h</code> below.

Table 5.10 Map Functions defined in `ct_map_utils.h`

Function	Description
<code>ct_create_user_map</code>	Creates a user map for user authentication. (Only verifies the user name.)
<code>ct_create_user_dn_map</code>	Creates a user map for user authentication with a distinguished name (dn).
<code>ct_create_user_basic_map</code>	Creates a user map for basic user authentication. (Contains user name and password.)
<code>ct_create_user_dn_basic_map</code>	Creates a user map for basic user authentication (user name and password) with a distinguished name rather than a simple user name.
<code>ct_create_user_ldap_map</code>	Creates a user map for user authentication (user name and password) on an LDAP system.
<code>ct_create_user_dn_ldap_map</code>	Creates a user map for user authentication (user name and password) with a distinguished name (dn) in an LDAP system.
<code>ct_create_user_securid_map</code>	Creates a user map for SecurID authentication (user name and pin + passcode).
<code>ct_create_user_securid_new_pin_map</code>	Creates a user map for SecurID authentication (user name and pin + passcode, where the pin is the new pin number).
<code>ct_create_user_securid_next_code_map</code>	Creates a user map for SecurID authentication (user name and pin + passcode, where passcode is the user's next passcode).
<code>ct_create_user_nt_map</code>	Creates a user map for NT user authentication (user name and password).
<code>ct_create_web_resource_map</code>	Creates a Web resource map required for authorizing access to a Web resource.
<code>ct_create_app_function_map</code>	For a specified application function (as defined in your RSA ClearTrust entitlements policies), creates the resource map required in order to authorize access to that function.
<code>ct_destroy_three_maps</code>	Destroys three maps. Often, when calling some of the <code>ct_runtime_api</code> (<code>ct_runtime_api.h</code>) functions, three maps are required. This replaces three calls to the <code>ct_map_destroy</code> function (<code>ct_map.h</code>) when the maps are no longer necessary.
<code>ct_destroy_two_maps</code>	Destroys two maps. Often, when calling some of the <code>ct_runtime_api</code> (<code>ct_runtime_api.h</code>) functions, two maps are required. This replaces two calls to the <code>ct_map_destroy</code> function (<code>ct_map.h</code>) when the maps are no longer necessary.
<code>ct_clear_three_maps</code>	Clears three maps. Often, when calling some of the <code>ct_runtime_api</code> (<code>ct_runtime_api.h</code>) functions, three maps are required. This replaces three calls to the <code>ct_map_clear</code> function (<code>ct_map.h</code>) when the maps' contents are no longer necessary.

Examples

RSA SecurID Authentication Example

This Runtime API example authenticates a user using the RSA SecurID method of authentication.

```
/*
 * securid_example.c
 *
 * Last updated February 12, 2002.
 */

#include "stdio.h"
#include "ct_runtime_api.h"
#include "ct_map_utils.h"
#include "ct_result_constants.h"

void print_map(const char* msg, ct_map* map)
{
    char map_string[1024];
    int map_len = 1024;
    ct_map_print(map, map_string, &map_len);
    printf(" %s : %s.\n" , msg, map_string);
}

void main(void)
{
    ct_pool_index pool_key ;
    char err_msg[1024];
    char username[20];
    char temp[20];
    char pass[20];

    boolean error = FALSE;
    ct_map* user_map = NULL;
    ct_map* result_map = ct_create_map();
    int i = 0;
    char* state = NULL;

    init_ct_runtime_api_with_ssl(TRUE);
    error = ct_create_pool_with_dispatcher(pool_key,
        TRUE, // round robin
        3, // retries
        10, // AS timeout
        "discovery", //dispatcher host
        "2608", //dispatcher port
        20, // dispatcher timeout
        FALSE, // SSL
        err_msg);
    printf(" Created pool from dispatcher.\n");
}
```

securid_example.c continues:

```

printf(" Input the User Name: ");
fgets(temp, 20, stdin);
sscanf(temp, "%s\n", username);

printf(" Input the user's pin+passcode: ");
fgets(temp, 20, stdin);
sscanf(temp, "%s\n", pass);

user_map = ct_create_user_securid_map(username, pass, state);
print_map(" The user map is ", user_map);
ct_authenticate_pool(pool_key,
                    user_map,
                    result_map);

if (user_map)
    ct_map_destroy(user_map);
print_map ("The result map is ", result_map);
if (!strcmp(ct_map_find(result_map, CT_AUTH_AUTHENTICATION_RESULT_STR),
            CT_AUTH_VALID_USER_STR))
{
    printf(" The authentication was successful. \n");
    state = NULL;
}
else if (!strcmp(ct_map_find(result_map,
                            CT_AUTH_AUTHENTICATION_RESULT_STR),
        CT_AUTH_SECURID_AUTH_FAILED_STR))
{
    printf(" The authentication failed. \n");
    state = NULL;
}
else if (!strcmp(ct_map_find(result_map,
                            CT_AUTH_AUTHENTICATION_RESULT_STR),
        CT_AUTH_NEW_PIN_REQUIRED_STR))
{
    char new_pin[10];
    ct_map * new_pin_map = NULL;

    char* server_key = strdup(ct_map_find(result_map,
                                        CT_AUTH_SERVER_KEY));

    printf(" You must create a new pin now.\n");
    printf(" Please type in your new pin:");
    fgets(temp, 6, stdin);
    sscanf(temp, "%s\n", new_pin);
    state = strdup(ct_map_find(result_map,
                              CT_AUTH_SERVER_STATE));
}

```

securid_example.c continues:

```
new_pin_map = ct_create_user_securid_new_pin_map(username,
                                                new_pin,
                                                state);

ct_map_clear(result_map);

ct_authenticate(server_key ,
                new_pin_map,
                result_map);
print_map (" The result map is ", result_map);
if (!strcmp(ct_map_find(result_map,
                        CT_AUTH_AUTHENTICATION_RESULT_STR),
          CT_AUTH_NEW_PIN_ACCEPTED_STR))
{
    ct_map* new_user_map = NULL;
    printf(" The new pin was accepted. \n");
    printf(" Please input pin+passcode: ");
    fgets(temp, 32, stdin);
    sscanf(temp, "%s\n", pass);
    state = strdup(ct_map_find(result_map,
                              CT_AUTH_SERVER_STATE));
    new_user_map = ct_create_user_securid_map(username,
                                              pass,
                                              state);

    print_map("The user map is ", new_user_map);
    ct_map_clear(result_map);
    ct_authenticate(server_key,
                    new_user_map,
                    result_map);
    print_map(" The result map is " ,
              result_map);
    if (new_user_map)
        ct_map_destroy(new_user_map);
    state = NULL;
}
else
{
    printf(" Sorry. The new pin was rejected.\n");
    state = NULL;
}

if (new_pin_map)
    ct_map_destroy(new_pin_map);
}
```

securid_example.c continues:

```
else if (!strcmp(ct_map_find(result_map,
                           CT_AUTH_AUTHENTICATION_RESULT_STR),
           CT_AUTH_NEXT_CODE_REQUIRED_STR))
{
    char next_code[6];
    char* server_key = strdup(ct_map_find(result_map,
                                         CT_AUTH_SERVER_KEY));

    ct_map* next_code_map = NULL;
    state = strdup(ct_map_find(result_map,
                              CT_AUTH_SERVER_STATE));
    printf(" Please input the next code on your token: ");
    fgets(temp, 32, stdin);
    sscanf(temp, "%s\n", next_code);
    next_code_map = ct_create_user_securid_next_code_map(username,
                                                         next_code, state);

    ct_map_clear(result_map);
    ct_authenticate(server_key,
                   next_code_map,
                   result_map);
    print_map (" The result map is ",
              result_map);
    state = NULL;
    if (next_code_map)
        ct_map_destroy (next_code_map);
}

if (result_map)
    ct_map_destroy(result_map);
}
```


6

Runtime Java API

The RSA ClearTrust® Runtime API allows trusted Java clients to perform authentication, authorization, and other functions using the runtime functionality of the RSA ClearTrust Authorization Servers. Using the information and examples in this chapter, together with the details provided in the RSA ClearTrust online Javadoc documentation, developers can build custom authentication or personalization programs that use or extend the RSA ClearTrust runtime functionality.

The RSA ClearTrust runtime methods are provided by the RuntimeAPI interface. RuntimeAPI and its supporting classes and interfaces are contained in the package `sirrus.runtime`.

Clients construct implementations of the RuntimeAPI interface using the static factory methods on the APIFactory class. These implementations are based on network communications with one or more RSA ClearTrust Authorization Servers.

An RSA ClearTrust Runtime API client may connect to the RSA ClearTrust Servers as an authenticated SSL client, an anonymous SSL client, or as a non-SSL client. Depending on your configuration, Runtime API applications may be required to connect over SSL in order to access and update user properties and SSO tokens. See “[Client Connection Options](#)” on page 162 for details.

See the RSA ClearTrust Runtime API Javadocs for precise descriptions of the Runtime API classes and methods. You can find the Javadocs in your RSA ClearTrust installation in the directory `<CT_HOME>/api/runtime-j/doc/index.html`.

This Chapter

This chapter consists of:

- an “[Overview](#)” of the Runtime API starting on page 158.
- compilation instructions starting with “[Installing and Compiling](#)” on page 161.
- reference information beginning with the “[Packages](#)” section on page 167.
- example programs with instructions:
 - “[Runtime API Example Without SSL](#)” on page 172
 - “[Runtime API Example With SSL](#)” on page 177
 - “[RSA SecurID Authentication Example](#)” on page 183

The source code for these API example programs is installed in

`<CT_HOME>/api/runtime-j/example`

Overview

What the Runtime API Does

The runtime functionality of the RSA ClearTrust Authorization Servers consists of:

- authentication of users.
- authorization of users to resources.
- SSO (single sign-on) token creation and manipulation.
- user property retrieval.

Authentication

Authentication, done by the `RuntimeAPI.authenticate()` method, confirms the identity of a user based on a set of credentials supplied by that user. The credentials required for a user authentication operation depend on the type of authentication being performed. You will specify the type of authentication required for a given resource (for example, a specific html page) on your network by setting the `cleartrust.agent.auth_resource_list` parameter in the RSA ClearTrust Web Server Agent's configuration file.

The Runtime API supports these authentication types:

- RSA ClearTrust basic authentication
- RSA SecurID authentication
- NT authentication
- LDAP authentication (Note that in RSA ClearTrust 4.7, this authentication type is equivalent to RSA ClearTrust basic authentication, since basic authentication verifies the user's password against an LDAP datastore. In future releases, basic authentication will verify the password against whatever type of underlying datastore you have configured to store users.)

In addition to the authentication types listed above, the API provides a mechanism to extend the Authorization Servers to include additional custom forms of user authentication.

For details on authentication, see `authenticate()` in Table 6.2 on page 168.



Note: Note that, in version 4.7, the Runtime API *does not* support Certificate Authentication. Previous versions of the Runtime API did support Certificate Authentication (and in 4.7, the *Web Server Agents* still do support Certificate Authentication; only the *API* does not).

Authorization

Authorization, done by the `RuntimeAPI.authorize()` method, checks whether a user has access to a specified resource. Authorization checks may be performed for a Web resource (a URL modeled as an `IApplicationURL`), an application (`IApplication`), or an application function (`IApplicationFunction`). When checking authorization, the Authorization Server invokes the RSA ClearTrust access control policy model, including basic entitlements and `SmartRules`. For details on the RSA ClearTrust policy model, see the *RSA ClearTrust Administrator's Guide 4.7*.

If desired, you can have the `authorize()` method perform user authentication as well as authorization. This is done by including an `AUTHENTICATION_TYPE` designation in the user credentials Map passed to `authorize()`.

For details on authorization, see `authorize()` in Table 6.2 on page 168.

SSO Token Manipulation

To maintain session state and to communicate authentication data between components, the Runtime API creates or updates a user's SSO (Single Sign-on) token when that user successfully authenticates. These tokens are passed between the Authorization Server, Web servers, application servers, and custom applications as a means of providing single sign-on functionality.

Since tokens contain sensitive session information, you may wish to allow only securely connected Runtime clients to create and manipulate tokens. The `cleartrust.runtime_api.security` setting in the Authorization Server's configuration file lets you do this. See "[Client Connection Options](#)" on page 162 for details.

Runtime API clients may instruct the server to implicitly return the user's token each time `authenticate()` or `authorize()` is called. To do this, the client must set the `SC_TOKENS_ENABLED` flag to `true` when creating the connection. The flag is set in the credentials Map passed when calling a create method in the `APIFactory` (for example, the `createAuthServerConnection()` method).

For information on using tokens, see `getTokenValue()` and related methods in Table 6.2 on page 168.

User Property Retrieval

Runtime API clients can view and update any user property value, as long as that user property is defined as exportable. User properties are optional fields that can be added to user records to store additional information about the user. Typically, user properties contain data to be evaluated by `SmartRules`.

User properties are created via the Administrative API or the RSA ClearTrust Entitlements Manager application. When the property is created, its exportable flag must be set to `true` if Runtime API clients are to see values stored in that property. Because some properties may be deemed too sensitive to externalize, only user properties which are defined to be exportable are visible to Runtime API clients.

Since user properties may contain confidential user information, you may wish to allow only securely connected Runtime clients to view and/or update user property

data. The `cleartrust.runtime_api.security` setting in the Authorization Server's configuration file lets you do this. See [“Client Connection Options”](#) on page 162 for details.

Runtime API clients may instruct the server to implicitly return the user's properties each time `authenticate()` or `authorize()` is called. To do this, the client must set the `SC_USER_PROPERTIES_ENABLED` flag to `true` when creating the connection. The flag is set in the credentials Map passed when calling a create method in the APIFactory (for example, the `createAuthServerConnection()` method). Because user property retrieval slows the execution of an authentication or authorization call, this flag defaults to `false`.

For information on user properties, see `getUserProperty()` in Table 6.2 on page 168. For information on creating user properties in your LDAP user data store, see the “Data Integrity” section in the “Installing the LDAP Data Adapters” chapter of the *RSA ClearTrust Installation and Configuration Guide 4.7*.

Runtime API Relies on Authorization Servers

A Runtime API client application that you build will connect to an Authorization Server to perform authentication, authorization, and most other tasks. The client application may connect to a single Authorization Server, or it may use an RSA ClearTrust Dispatcher to build a pool of connections to many Authorization Servers. Connection pools automatically and transparently fail over requests in the event of an Authorization Server crash or network outage, and can be configured to distribute requests across available servers in the pool in a round-robin fashion.

Runtime API Calls Are Threadsafe

RuntimeAPI methods are fully thread-safe (i.e. a single RuntimeAPI object can service requests from multiple threads with no need for additional synchronization code).

Runtime API vs. Administrative API

The purpose of a Runtime API client is to respond to a user's request by authenticating the user and authorizing him or her to use a ClearTrust-protected resource. Administrative API clients, on the other hand, are used to update users' access permissions to ClearTrust-protected resources and, optionally, to maintain user records.

The Runtime API differs from the Administrative API in that runtime tasks are *read-only*; that is, they perform queries on the existing state of the data in the RSA ClearTrust Servers, but do not involve changing or adding data. To perform tasks involving updates, you must use the RSA ClearTrust Administrative API, as explained in [Chapter 4, “Administrative Java API”](#). Both APIs may be used in a single client program.

Because a Runtime API client application will not make changes to RSA ClearTrust data, it does not need to provide a ClearTrust administrator ID and password at startup. Administrative API applications must always log in with an ID and password.

Runtime client applications differ further in that they connect to Authorization Servers, while Administrative API applications connect to the RSA ClearTrust API Server, which is part of the Entitlements Server.

Installing and Compiling

This section explains the installed components that make up the API and provides guidelines for building applications. For instructions on installing the APIs, see [Chapter 2, “Installing the RSA ClearTrust APIs”](#).

Compiling Applications

In order to compile and run API programs, you must have the Runtime API jar, `ct_runtime_api.jar`, installed and included in your `SOURCEPATH` and `CLASSPATH`. You will find this jar file installed as:

```
<CT_HOME>/api/runtime-j/lib/ct_runtime_api.jar
```

In addition, if your Runtime API client program will connect over SSL (“`cleartrust.api.server.use_ssl=true`” and related settings), you will also need following jar files in your `SOURCEPATH` and `CLASSPATH`. You will find these jars in your RSA ClearTrust installation in the `<CT_HOME>/lib` directory.

RSA SSL software:

- `certj.jar`
- `jsafe.jar`
- `jsafeJCE.jar`
- `rsajsse.jar`
- `sslj.jar`

JCSI Keystore software:

- `jcsi_base.jar`
- `jcsi_provider.jar`

Sun security infrastructure:

- `jce1_2-do.jar`
- `jcet.jar`
- `jnet.jar`
- `jsse.jar`

Client Connection Options

An RSA ClearTrust Java Runtime API client may connect as an authenticated SSL client, an anonymous SSL client, or as a clear text (non-SSL) client. Authenticated clients communicate with the RSA ClearTrust Servers via a mutually authenticated, encrypted SSL connection. Anonymous SSL clients are not authenticated but communicate over an encrypted SSL connection. Non-SSL clients communicate without encryption, in cleartext.

This section explains the cases in which you may wish to require SSL connections and how to connect using the three approaches.

Access to Tokens and User Properties

In order to protect the sometimes sensitive information in SSO tokens and user properties, RSA ClearTrust can be configured to check that a Runtime API client has established a secure enough connection before returning a token or property value.

This configuration is done by setting the `cleartrust.runtime_api.security` parameter in your Authorization Server's configuration file (`aserver.conf`). This parameter sets the minimum security required on the connection between the Runtime API client and the Authorization Server in order to create/manipulate SSO tokens and to retrieve user properties. Valid settings, in order of increasing security, are:

- `cleartext` for a minimum connection type of *clear text*. This allows tokens/properties to be passed over any type of connection.
- `anonymous` for a minimum connection type of *anonymous SSL*. This allows tokens/properties to be passed over anonymous or authenticated SSL connections.
- `authenticated` for a minimum connection type of *authenticated SSL*. This is the default. This allows tokens/properties to be passed over authenticated SSL connections only.

The following table summarizes the available levels of required security for connections. In this table, a “Yes” indicates that the Runtime API client of that row will be permitted to view tokens/user properties by an Authorization Server

configured with the `cleartrust.runtime_api.security` setting noted in that column.

Table 6.1 Access matrix for Runtime API clients wishing to access tokens/user properties

			Level of security required by Authorization Server (per setting of <code>cleartrust.runtime_api.security</code>)		
			cleartext	anonymous	authenticated
Runtime API client's connection type	cleartext	<code>cleartrust.net.ssl.use=NO</code> <code>cleartrust.net.ssl.require_authentication=false</code>	Yes	No	No
	anonymous	<code>cleartrust.net.ssl.use=YES</code> <code>cleartrust.net.ssl.require_authentication=false</code>	Yes	Yes	No
	authenticated	<code>cleartrust.net.ssl.use=YES</code> <code>cleartrust.net.ssl.require_authentication=true</code>	Yes	Yes	Yes

Token Access Example: No SSL

If the Authorization Server is configured with no SSL (in the `aserver.conf`, set `cleartrust.net.use.ssl=No`) and you set

```
cleartrust.runtime_api.security=anonymous
```

or

```
cleartrust.runtime_api.security=authenticated
```

then all Runtime API clients connecting to this Authorization Server will *not* be able to create/manipulate tokens or retrieve user properties. Only with this parameter set to `cleartext` will your system permit Runtime API clients to access tokens and user properties.

Token Access Example: Authenticated SSL

If the Authorization Server is configured to use authenticated SSL (the most secure type of connection), then, regardless of how you set `cleartrust.runtime_api.security`, the Runtime API clients *will* be able to create/manipulate SSO tokens and retrieve user properties.

Connecting Over Authenticated SSL

With an Authenticated SSL connection, the client will exchange credentials with the RSA ClearTrust Servers so that each party may authenticate the other, and then the client and servers will communicate over an encrypted SSL connection.

In order to require that Runtime API clients connect in this way, you must make the following settings in the Authorization Server's `aserver.conf` file:

```
clearTrust.net.ssl.use=yes
cleartrust.net.ssl.require_authentication=yes
```

In order to connect over authenticated SSL, the client program needs to load its cryptography and certificate providers, load a keystore from a keystore file, and pass the keystore to the `APIFactory` to create and connect the authenticated `RuntimeAPI` client session. The following sections explain these steps.

Add a Security Provider

To accomplish the first step, adding the security provider, call the following methods from the `sirrus.util.crypt.SecurityProviderLoader` class before you read in your keystore. The security provider handles key generation, conversion, and management.

```
SecurityProviderLoader.loadCryptoProviders();
SecurityProviderLoader.loadCertificateProviders();
```

Create Keystore

Next, you can create the keystore by reading in the contents of the client keystore file. You must provide a passphrase to unlock the keystore, and you must have access to the valid client keystore file for this client. To simplify the keystore-loading calls, we will define the following constants:

```
KeyStore clientKeyStore = null;
String keystoreFile = "c:\\temp\\kca.p12";
String keystorePassword = "abc123";
char[] myKeystorePhrase = keystorePassword.toCharArray();
```


The keystore can be loaded as follows:

```
try
{
    SecurityProviderLoader.loadCryptoProviders();
    SecurityProviderLoader.loadCertificateProviders();

    clientKeyStore = KeyStore.getInstance("PKCS12", "DSTC_PKCS12");
    clientKeyStore.load(new FileInputStream(keystoreFile), myKeystorePhrase);
}
catch (Exception e)
{
    System.out.println("Exception in Keystore.");
    e.printStackTrace();
    return;
}
```



Note: The application's search for the keystore file depends on whether the CT_ROOT environment variable has been set. If CT_ROOT is defined, then the Runtime API application assumes the keystore path is relative to <CT_ROOT>/conf. If no file is found at this location, then the application assumes the path is relative to the present working directory and looks there. If CT_ROOT is *not* defined, then the application looks for the file relative to the present working directory. (The present working directory is the directory where the Java session was launched.)

Add Keystore to Credentials Map and Connect

Finally, you can connect as follows:

- Create a credentials Map containing the keystore and passphrase (in the SC_SSL_KEYSTORE and SC_SSL_PRIVATE_KEY_PASSPHRASE fields, respectively).
- Create a ServerDescriptor with the SSL setting (here called “useSSL”), the name of the Dispatcher machine, and the port of the Dispatcher.
- Create and connect the authenticated SSL session by providing the credentials Map and ServerDescriptor to the APIFactory.

The RuntimeAPI object should be declared in the main body of your class:

```
private RuntimeAPI runtimeAPI= null;
```

To simplify the connection calls, we will define the following constants:

```
ServerDescriptor dispatcher = null;
String dispatchServer = "localhost";
int dispatchServerPort = 5608;
boolean useSSL = true;
Map runtimeCredentials = new HashMap();
```

The following lines show how you create the credentials, create the ServerDescriptor, and connect.

```
runtimeCredentials.put( CredentialConstants.SC_SSL_KEYSTORE,
                        clientKeyStore );
runtimeCredentials.put( CredentialConstants.SC_SSL_PRIVATE_KEY_PASSPHRASE,
                        myKeystorePhrase );

dispatcher = new ServerDescriptor(dispatchServer,
                                  dispatchServerPort,
                                  useSSL);

try
{
    runtimeAPI =
        APIFactory.createFromServerDispatcher(runtimeCredentials,
                                              dispatcher);
}
catch( RuntimeAPIException e )
{
    System.out.println("Error connecting.");
    e.printStackTrace();
}
```

The code segments above are taken from the complete example program beginning on page 177.

For information on setting up your RSA ClearTrust Servers for SSL communication, consult the section “Configuring RSA ClearTrust to Use Authenticated SSL” in the *RSA ClearTrust Installation and Configuration Guide 4.7*.

Connecting Over Anonymous SSL

Anonymous SSL clients are not authenticated but communicate over an encrypted SSL connection.

In order to require that Runtime API clients connect in this way, you must make the following settings in the Authorization Server's `aserver.conf` file:

```
cleartrust.net.ssl.use=YES
cleartrust.net.ssl.require_authentication=false
cleartrust.runtime_api.security=anonymous
```

Connecting Without SSL

Non-SSL clients communicate without encryption.

In order to require that Runtime API clients connect in this way, you must make the following settings in the Authorization Server's `aserver.conf` file:

```
cleartrust.net.ssl.use=NO
cleartrust.net.ssl.require_authentication=false
cleartrust.runtime_api.security=cleartext
```

Packages

The Runtime API consists of two packages:

- `sirrus.runtime` — API to the runtime functionality of the RSA ClearTrust system, as provided by the Authorization Servers.
- `sirrus.runtime.customauth` — mechanism for adding custom authentication types to the RSA ClearTrust system.

All the core methods of the API are contained in the `sirrus.runtime` package. The `RuntimeAPI` interface describes these methods, and the `APIFactory` class provides methods for instantiating objects that implement `RuntimeAPI`. All methods are fully thread-safe (i.e., a single `RuntimeAPI` object can service requests from multiple threads with no need for additional synchronization code).

The `sirrus.runtime.customauth` package allows you to build applications that implement custom forms of user authentication. This package is not described further in this document. Consult the online Javadoc documentation for details.

Interfaces

This section describes the interfaces provided by the Runtime API.

Interface `RuntimeAPI`

The `RuntimeAPI` interface provides nearly all of the methods that a Runtime API client will use. To begin using the `RuntimeAPI` methods, the client application will first use one of the `APIFactory` methods to create a `RuntimeAPI`-implementing object that is connected to an Authorization Server or to a pool of Authorization Servers. The `createAuthServerConnection` methods create a `RuntimeAPI` object connected to one Authorization Server, and the `createFromServerDispatcher` methods and `createFromServerList` methods create a `RuntimeAPI` object that is connected to a pool of Authorization Servers. A pool of connections provides failover and, if desired, load-balancing.

In addition to the main runtime functions described in “[What the Runtime API Does](#)” on page 158, a `RuntimeAPI` object provides a number of utility functions:

- `checkResourceStatus()` to check whether a given resource is protected;
- `clearServerCaches()` to force the Authorization Server(s) to clear their caches
- `testServer()` to “ping” the Authorization Server(s);
- and `close()` to close all of the Runtime API client application’s connections to the Authorization Server(s).

See Table 6.2 for a complete list of `RuntimeAPI` methods.

The arguments and returned values of many of the `RuntimeAPI` methods have the type `java.util.Map`, an interface in the standard Java collections API. Each such map is expected to contain one or more key-value pairs describing the argument or result in question.

The keys and possible values for maps representing users, user credentials, and resources are documented in `UserConstants` (see page 169), `CredentialConstants` (see page 170), and `ResourceConstants` (see page 170). `ResultConstants` (see page 170) describes what keys and values may be expected in maps returned by the API methods. The Javadoc comments for the `RuntimeAPI` methods contains additional details on the possible return values for each call.

All methods except `testServer()` and `close()` may throw a `RuntimeAPIException`. This indicates a catastrophic error in servicing the request; for example, an Authorization Server connection could not be established, or the connection has already been closed with the `close()` method. Note that in the case that this instance of `RuntimeAPI` is backed by an Authorization Server pool (that is, the instance was obtained by calling `APIFactory.createFromServerDispatcher()`, `APIFactory.createFromServerDispatchers()`, or `APIFactory.createFromServerList()`), a `RuntimeAPIException` can occur only after all servers in the pool have been tried, and each has failed to service the request.

The following table describes the `RuntimeAPI` interface methods.

Table 6.2 Interface `RuntimeAPI` Methods

Method	Description
<code>authenticate</code>	Authenticates a user.
<code>authorize</code>	Authorizes a user to a resource.
<code>createToken</code>	Creates a new SSO token for a user.
<code>getTokenValue</code>	Get the value for a particular key in the given token.
<code>getTokenValues</code>	Get all the keys and values in the given token.
<code>setTokenValue</code>	Set a value for a key in the given token, and return the resulting new token.
<code>setTokenValues</code>	Set the values of multiple keys in the given token, and return the resulting new token.
<code>getUserProperty</code>	Return the value of the given property for the user.
<code>getUserProperties</code>	Returns the values of all exportable user properties for a user.
<code>testServer</code>	Pings the server that implements this interface.
<code>checkResourceStatus</code>	Checks the protection status of a given resource.
<code>clearServerCaches</code>	Clears the server cache of all the Authorization Servers to which this Runtime client is connected. If this client is backed by an Authorization Server pool, the caches of all servers in the pool will be cleared.
<code>close</code>	Closes all connections to all the Authorization Server(s) backing this Runtime client.

Interface UserConstants

This class contains constant Strings to be used as keys or values in Map objects representing RSA ClearTrust users and their authentication credentials in calls to the `RuntimeAPI`.

All user Maps must contain a value for `SC_USER_ID`, `SC_USER_DN`, `SC_USER_CERT`, or `SC_TOKEN`, but NOT more than one of these. Whichever value is present will determine how the user is identified in the RSA ClearTrust system: by user ID, by distinguished name, by the DN contained in a certificate, or by the ID contained in an SSO token, respectively. In addition, for calls to `authenticate()`, a value for `AUTHENTICATION_TYPE` must be present; this key-value pair may also be optionally provided in calls to `authorize()`, forcing the `authorize()` call to first authenticate the user.

Interface TokenKeys

This class contains constant Strings that may be used as keys in an RSA ClearTrust SSO token. A token is an encrypted String which encodes a set of key-value pairs. Values are associated with keys in a token using the `RuntimeAPI.setTokenValue()` method, and can be retrieved using `RuntimeAPI.getTokenValue()`.

In addition to the keys listed here, the Strings denoting authentication types listed in `AuthTypes` are also valid keys in an SSO token. The value associated with an auth type string will be either `true` or `false`, indicating whether or not authentication of that type has been successfully performed.



Note: In this version of the API, only these keys and the ones from `AuthTypes` may be used in a token. Clients are responsible for encoding and decoding any custom key-value pairs they may need into a value for the key `TokenKeys.SC_CUSTOM_DATA`.

Interface AuthTypes

This interface contains a small set of constant strings which denote the various types of authentication supported in the RSA ClearTrust system. These strings are used as follows:

- as values for `UserConstants.AUTHENTICATION_TYPE` in user Maps passed to `Runtime.authenticate()` or `Runtime.authorize()`
- as keys in an RSA ClearTrust SSO token.

In a token, one or more of these strings may be associated with the value “true”, indicating that the corresponding type of authentication has succeeded. Any other value indicates that the authentication type either hasn't been attempted, or was attempted and failed.

If custom authentication types are deployed, their identifying strings should not begin with the prefix “SC_”, to ensure uniqueness.

Interface ResourceConstants

This interface defines the constant Strings to be used as keys and values in a Map object representing an RSA ClearTrust resource (either a URL or an application function) in calls to the `RuntimeAPI`.

When you create a Map for a URL, its `TYPE` will be `WEB_RESOURCE` and it will have a `WEB_SERVER_NAME` and a URI string. For example, you could build a map for the URL, “/myPage.html”, as follows:

```
Map myResource = new HashMap();
myResource.put(ResourceConstants.TYPE, ResourceConstants.WEB_RESOURCE);
myResource.put(ResourceConstants.WEB_SERVER_NAME, "myApacheServer");
myResource.put(ResourceConstants.URI, "/myPage.html");
```

When you create a Map for an application function, its `TYPE` will be `APP_FUNCTION` and it will have an `APPLICATION_NAME` and a `FUNCTION_NAME`.

Interface ResultConstants

This interface defines the constant Strings that are used as keys or values in Map objects returned by calls to `RuntimeAPI` methods. Most result Maps must have a value for the key `RETURN_CODE` and may have additional key-value pair(s) depending on the context. See the Javadoc documentation of each method in `RuntimeAPI` for a list of the `ResultConstants` that can be returned by that method.

Interface CredentialConstants

This interface defines the constant strings used to supply client authentication credentials to the `APIFactory` when creating a `RuntimeAPI` object. For SSL-connected clients, you must supply the client application's keystore in the `SC_SSL_KEYSTORE` key and the password for the client's key in the `SC_SSL_PRIVATE_KEY_PASSPHRASE` key.

To turn on automatic retrieval of users' SSO tokens, set the `SC_TOKENS_ENABLED` key to true. To turn on automatic retrieval of user properties, set the `SC_USER_PROPERTIES_ENABLED` key to true.

Runtime API Classes

This section summarizes the Runtime API classes.

Class APIFactory

The `APIFactory` class extends `java.lang.Object`.

This class provides a small set of static methods which construct and return objects which implement the `RuntimeAPI` interface. These factory methods provide the starting point for all Runtime API client applications.

The simplest factory method is `createAuthServerConnection()`, which opens and returns a single connection to a single Authorization Server. The other methods each build a pool of Authorization Server proxies with some connection management and failover capability built in. The differences among these methods lie in how the servers in the pool are looked up: via the RSA ClearTrust Dispatcher, or from a `List` of `ServerDescriptors`. In either case, the server lookup source (Dispatcher or List) will be re-read as necessary to refresh the contents of the pool.

Table 6.3 Class APIFactory Methods

Method	Description
<code>createAuthServerConnection</code>	Creates a single connection to an Authorization Server.
<code>createFromServerDispatcher</code>	Creates a server pool from a ClearTrust Dispatcher.
<code>createFromServerDispatchers</code>	Creates a server pool from available redundant ClearTrust Dispatchers.
<code>createFromServerList</code>	Creates a server pool from a list of <code>ServerDescriptor</code> objects.
<code>getClearTrustVersion</code>	Gets a string describing the current version of ClearTrust.

Class ServerDescriptor

Describes where a server is running and how to connect to it. Instances of this class are just “dumb data” objects encapsulating a host address, a port number, and a connection type (SSL or straight TCP). This information is what is needed for connecting to RSA ClearTrust Authorization Servers and Server Dispatchers.

Table 6.4 Class ServerDescriptor Methods

Method	Description
<code>ServerDescriptor</code>	Constructor to build a descriptor for a server at the given address and port with SSL on or off.
<code>getHost</code>	Gets this server's host.
<code>getPort</code>	Gets this server's port.
<code>isSSLused</code>	Returns a boolean indicating the connection mode of this server.

Examples

Runtime API Example Without SSL

This Runtime API example, “`RuntimeExample`,” authenticates a user and checks whether that user is authorized to read the specified URL. Specifically, this application constructs a server pool that gives it access to one or more Authorization Servers (based on an RSA ClearTrust Dispatcher), makes authentication and authorization requests against that pool, and finally disconnects. This example works only in a non-SSL installation of RSA ClearTrust; for the SSL version of this example, see page [177](#).

Before You Run the Program

Before compiling and running this program, you must do the following:

1. Edit the `dispatchServer` variable to the name of the machine where your Dispatcher is running.
2. Make sure you have a user, a Web server, and a URI saved in your RSA ClearTrust Policy data store.

For instructions on compiling examples, see “[Compiling Applications](#)” on page [161](#).

Example

```
package sirrus.samples.runtime;

import java.io.*;
import java.util.*;
import java.security.KeyStore;
import java.util.HashMap;
import sirrus.runtime.*;

/**
 * RuntimeExample.java
 *
 * @version 4.7
 * @since October 19, 2001
 */
public class RuntimeExample
{
    private RuntimeAPI runtimeAPI= null;

    private void connect()
    {
        ServerDescriptor dispatcher = null;
        String dispatchServer = "localhost";
        int dispatchServerPort = 5608;
        boolean useSSL = false;

        dispatcher = new ServerDescriptor(dispatchServer,
                                         dispatchServerPort,
                                         useSSL);

        try{
            runtimeAPI = APIFactory.createFromServerDispatcher(dispatcher);
        }
        catch( RuntimeAPIException e ){
            System.out.println("Error connecting.");
            e.printStackTrace();
        }
    }

    private void disconnect()
    {
        if (runtimeAPI != null)
            runtimeAPI.close();
    }
}
```

RuntimeExample continues:

```
private void authenticateUser(String userID, String password)
{
    String authcResult = null;
    String returnCode = null;
    Map result = null;
    Map user = new HashMap();

    user.put( UserConstants.SC_USER_ID, userID);
    user.put( UserConstants.AUTHENTICATION_TYPE, AuthTypes.SC_BASIC );
    user.put( UserConstants.CREDENTIALS, password );

    System.out.println("Checking user. Please wait...");

    try{
        result = runtimeAPI.authenticate( user );

        authcResult = (String)result.get( ResultConstants.AUTHENTICATION_RESULT);
        System.out.println("authentication_result = " + authcResult );

        returnCode = (String) result.get( ResultConstants.RETURN_CODE);
        System.out.println("Authentication return_code = " + returnCode );

    }catch(RuntimeAPIException e){
        e.printStackTrace();
    }

    return;
}

private void authorizeUser(String userID, String wsname, String url)
{
    String returnCode = null;
    Map result = null;
    Map user = new HashMap();
    Map resource = new HashMap();

    user.put( UserConstants.SC_USER_ID, userID);

    resource.put( ResourceConstants.TYPE, ResourceConstants.WEB_RESOURCE );
    resource.put( ResourceConstants.WEB_SERVER_NAME, wsname );
    resource.put( ResourceConstants.URI, url );

    System.out.println("Checking user access. Please wait...");
}
```

RuntimeExample continues:

```
try{
    result = runtimeAPI.authorize( user, resource );

    returnCode = (String) result.get( ResultConstants.RETURN_CODE );
    System.out.println("Authorization return_code = " + returnCode );

}catch(RuntimeAPIException e){
    e.printStackTrace();
}

return;
}

public static void main(String[] args)
{
    RuntimeExample apiClient = new RuntimeExample();

    // Initialize connection to ClearTrust
    apiClient.connect();

    // Get first user-entered string
    System.out.println("User name?");
    BufferedReader br1 =
        new BufferedReader(new InputStreamReader(System.in));
    String nameIn = new String("0");
    try{
        nameIn = br1.readLine();
    }catch(Exception e){
        e.printStackTrace();
    }

    // Get second user-entered string
    System.out.println("Password?");
    BufferedReader br2 =
        new BufferedReader(new InputStreamReader(System.in));
    String pswdIn = new String("0");
    try{
        pswdIn = br2.readLine();
    }catch(Exception e){
        e.printStackTrace();
    }
}
```

RuntimeExample continues:

```
// Get third user-entered string
System.out.println("Web server name?");
BufferedReader br3 =
    new BufferedReader(new InputStreamReader(System.in));
String wsnameIn = new String("0");
try{
    wsnameIn = br3.readLine();
}catch(Exception e){
    e.printStackTrace();
}

// Get fourth user-entered string
System.out.println("URL address?");
BufferedReader br4 =
    new BufferedReader(new InputStreamReader(System.in));
String urlIn = new String("0");
try{
    urlIn = br4.readLine();
}catch(Exception e){
    e.printStackTrace();
}

// Try to authenticate the user
apiClient.authenticateUser(nameIn, pswdIn);

// Try to authorize the user
apiClient.authorizeUser(nameIn, wsnameIn, urlIn);

// Disconnect from ClearTrust
apiClient.disconnect();
}
}
```

Runtime API Example With SSL

This Runtime API example, “`RuntimeSSLExample`,” connects over SSL and attempts to authenticate and authorize a user. Specifically, this application constructs a server pool that gives it access to one or more Authorization Servers (based on an RSA ClearTrust Dispatcher), makes authentication and authorization requests against that pool, and finally disconnects.

This example works only in an SSL-enabled installation of RSA ClearTrust; for the non-SSL version of this example, see page [172](#).

Before You Run the Program

Before this program can connect over authenticated SSL, you must

1. Make sure a client-side keystore file is available where the API client will run. Details for setting up the keystore can be found in the *RSA ClearTrust Installation and Configuration Guide 4.7*, in the section, “Configuring RSA ClearTrust to Use Authenticated SSL (PKI).”
2. Edit the program to set the `keystoreFile` variable to the path name of the client-side keystore file and add the `keystorePassword` for this keystore.
3. Edit the `dispatchServer` variable to the name of the machine where your Dispatcher is running.
4. Make sure the rest of your RSA ClearTrust installation is running in SSL mode. This requires that the `cleartrust.net.ssl.*` parameters be set in your `eserver.conf`, `dispatcher.conf`, and `aserver.conf` files, and that each RSA ClearTrust component have a keystore file. As with the client-side keystore above, see the *RSA ClearTrust Installation and Configuration Guide 4.7* for details on setting up keystores.
5. Make sure you have a user, a Web server, and a URI saved in your RSA ClearTrust Policy data store.

For instructions on compiling examples, see “[Compiling Applications](#)” on page [161](#).

Example

```
package sirrus.samples.runtime;

import java.io.*;
import java.util.*;
import java.security.KeyStore;
import java.util.HashMap;
import sirrus.runtime.*;
import sirrus.util.crypt.SecurityProviderLoader;

/**
 * RuntimeSSEExample.java
 *
 * @version 4.7
 * @since October, 2001
 */
public class RuntimeSSEExample
{
    private static RuntimeAPI runtimeAPI= null;

    private void connect()
    {
        ServerDescriptor dispatcher = null;
        String dispatchServer = "localhost";
        int dispatchServerPort = 5608;
        boolean useSSL = true;
        Map runtimeCredentials = new HashMap();
        KeyStore clientKeyStore = null;
        String keystoreFile = "c:\\temp\\kca.p12";
        String keystorePassword = "abc123";
        char[] myKeystorePhrase = keystorePassword.toCharArray();

        // Load the keystore for this client application.
        try
        {
            SecurityProviderLoader.loadCryptoProviders();
            SecurityProviderLoader.loadCertificateProviders();

            clientKeyStore = KeyStore.getInstance( "PKCS12", "DSTC_PKCS12" );
            clientKeyStore.load( new FileInputStream( keystoreFile ),
                                myKeystorePhrase );
        }
        catch (Exception e)
        {
            System.out.println("Exception in Keystore.");
            e.printStackTrace();
            return;
        }
    }
}
```

RuntimeSSLExample continues:

```
// Connect to the API Server.
runtimeCredentials.put( CredentialConstants.SC_SSL_KEYSTORE,
                        clientKeyStore );
runtimeCredentials.put( CredentialConstants.SC_SSL_PRIVATE_KEY_PASSPHRASE,
                        myKeystorePhrase );

dispatcher = new ServerDescriptor(dispatchServer,
                                  dispatchServerPort,
                                  useSSL);

try{
    runtimeAPI =
        APIFactory.createFromServerDispatcher(runtimeCredentials,
                                             dispatcher);
}
catch( RuntimeAPIException e ){
    System.out.println("Error connecting.");
    e.printStackTrace();
}

}

private void disconnect()
{
    if (runtimeAPI != null)
        runtimeAPI.close();
}

private void authenticateUser(String userID, String password)
{
    String authResult = null;
    String returnCode = null;
    Map result = null;
    Map user = new HashMap();

    user.put( UserConstants.SC_USER_ID, userID);
    user.put( UserConstants.AUTHENTICATION_TYPE, AuthTypes.SC_BASIC );
    user.put( UserConstants.CREDENTIALS, password );

    System.out.println("\nChecking user. Please wait...");

    try{
        result = runtimeAPI.authenticate( user );

        authResult = (String)result.get(ResultConstants.AUTHENTICATION_RESULT);
        System.out.println("authentication_result = " + authResult );

        returnCode = (String) result.get(ResultConstants.RETURN_CODE);
        System.out.println("return_code = " + returnCode );
    }
}
```

RuntimeSSLExample continues:

```

    }catch(RuntimeAPIException e){
        e.printStackTrace();
    }

    return;
}

private void authorizeUser(String userID, String wsname, String url)
{
    String returnCode = null;
    Map result = null;
    Map user = new HashMap();
    Map resource = new HashMap();

    user.put( UserConstants.SC_USER_ID, userID);

    resource.put( ResourceConstants.TYPE, ResourceConstants.WEB_RESOURCE );
    resource.put( ResourceConstants.WEB_SERVER_NAME, wsname );
    resource.put( ResourceConstants.URI, url );

    System.out.println("Checking user access. Please wait...");

    try{
        result = runtimeAPI.authorize( user, resource );

        returnCode = (String) result.get( ResultConstants.RETURN_CODE);
        System.out.println("Authorization return_code = " + returnCode );

    }catch(RuntimeAPIException e){
        e.printStackTrace();
    }

    return;
}

public static void main(String[] args)
{
    RuntimeSSLExample apiClient = new RuntimeSSLExample();

    // Initialize connection to ClearTrust API Server
    apiClient.connect();

    // Test server connection
    Map testResult = new HashMap();
    testResult = runtimeAPI.testServer(null);
    System.out.println("\n" + testResult.toString() + "\n");
}

```


RuntimeSSLExample continues:

```
// Get first user-entered string
System.out.println("User name?");
BufferedReader br1 =
    new BufferedReader(new InputStreamReader(System.in));
String nameIn = new String("0");
try{
    nameIn = br1.readLine();
}catch(Exception e){
    e.printStackTrace();
}

// Get second user-entered string
System.out.println("Password?");
BufferedReader br2 =
    new BufferedReader(new InputStreamReader(System.in));
String pswdIn = new String("0");
try{
    pswdIn = br2.readLine();
}catch(Exception e){
    e.printStackTrace();
}

// Get third user-entered string
System.out.println("Web server name?");
BufferedReader br3 =
    new BufferedReader(new InputStreamReader(System.in));
String wsnameIn = new String("0");
try{
    wsnameIn = br3.readLine();
}catch(Exception e){
    e.printStackTrace();
}

// Get fourth user-entered string
System.out.println("URL address?");
BufferedReader br4 =
    new BufferedReader(new InputStreamReader(System.in));
String urlIn = new String("0");
try{
    urlIn = br4.readLine();
}catch(Exception e){
    e.printStackTrace();
}
```

RuntimeSSLExample continues:

```
// Try to authenticate the user
apiClient.authenticateUser(nameIn, pswdIn);

// Try to authorize the user
apiClient.authorizeUser(nameIn, wsnameIn, urlIn);

// Disconnect from API Server
apiClient.disconnect();
}
}
```

RSA SecurID Authentication Example

This Runtime API example authenticates a user using the RSA SecurID method of authentication. For instructions on compiling examples, see [“Compiling Applications”](#) on page 161.

```
package sirrus.samples.runtime;

import sirrus.runtime.*;
import sirrus.util.crypt.SecurityProviderLoader;
import java.util.*;
import java.math.*;
import java.io.*;
import java.security.*;
import java.security.cert.*;

/**
 * SecurIdExample.java
 *
 * @version 4.7
 * @since November 9, 2001
 */
public class SecurIdExample
{
    static
    {
        SecurityProviderLoader.loadCryptoProviders();
        SecurityProviderLoader.loadCertificateProviders();
    }

    String strKeyFile = "c:\\temp\\kca.p12";
    char[] charStorePassPhrase = {'a', 'b', 'c', '1', '2', '3'};
    char[] charKeyPassPhrase = {'a', 'b', 'c', '1', '2', '3'};

    String authType = "sd";
    BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

    public static void main(String args[])
    {
        SecurIdExample ts = new SecurIdExample();
        int loop = 1;
        boolean ssl = false;
        RuntimeAPI api = null;
        HashMap user = new HashMap();

        try
        {
            if ( args.length == 2 )
            {
                loop = Integer.parseInt( args[0] );
                ssl = Boolean.valueOf( args[1] ).booleanValue();
            }
        }
    }
}
```

SecurIdExample continues:

```

        api = ts.createAPI( ssl );

        for (int i = 0; i < loop; i++ )
            {
                ts.readInformation( user );
                ts.sendRequest( api, user );
            }

        ts.closeInput();
    }
    catch( Exception e )
    {
        e.printStackTrace();
    }

    if (api != null)
    {
        api.close();
    }
}

void readInformation( Map user ) throws Exception
{
    System.out.println("Input your data in the " +
        "order \" username passcode \" ");
    String info = readLine();
    StringTokenizer sto = new StringTokenizer( info );

    while (sto.countTokens() != 2 )
    {
        System.out.println( " You did not enter your data in the " +
            "correct format, please re-type it. " );
        info = readLine();
        sto = new StringTokenizer( info );
    }

    String username = sto.nextToken();
    String passcode = sto.nextToken();

    user.clear();
    putInformation( user, username, passcode );
}

RuntimeAPI createAPI( boolean ssl )
{
    RuntimeAPI api = null;

```

SecurIdExample continues:

```
try
{
    ServerDescriptor dispatchers[] =
    { new ServerDescriptor("localhost", 5608, ssl) };

    String[] prefs = {"CLASS0", "CLASS1"};
    Map credentials = new HashMap();
    Map user = new HashMap();

    if (ssl == true)
    {
        KeyStore ks = KeyStore.getInstance("PKCS12", "DSTC_PKCS12");
        ks.load(new FileInputStream(strKeyFile),
                charStorePassPhrase);
        credentials.put(CredentialConstants.SC_TOKENS_ENABLED,
                        "true");
        credentials.put(CredentialConstants.SC_SSL_KEYSTORE, ks);
        credentials.put(
            CredentialConstants.SC_USER_PROPERTIES_ENABLED,
            "false");
        credentials.put(
            CredentialConstants.SC_SSL_PRIVATE_KEY_PASSPHRASE,
            charKeyPassPhrase);
    }

    api = APIFactory.createFromServerDispatchers( credentials,
                                                dispatchers,
                                                120000,
                                                120000,
                                                true,
                                                prefs );
}
catch( Exception e )
{
    e.printStackTrace();
}

return api;
}

String readLine() throws Exception
{
    String line = null;

    line = br.readLine();

    return line;
}
```

SecurIdExample continues:

```

void closeInput() throws Exception
{
    br.close();
}

void sendRequest( RuntimeAPI api, Map user )
    throws Exception
{
    Map result = new HashMap();
    String username = ( String )user.get( UserConstants.SC_USER_ID );
    result = api.authenticate( user );
    System.out.println(" The SC_AUTH_STATE is : " +
        result.get( ResultConstants.SC_AUTH_STATE ) );

    //String token =(String) result.get(UserConstants.SC_TOKEN);
    //Map ret = api.getTokenValues(token);
    //System.out.println(ret);

    if (result == null )
    {
        System.out.println( " No result was returned. " +
            "Check your ACE Server set-up." );
        return;
    } // end of if (result == null )

    String code = (String)result.get(ResultConstants.AUTHENTICATION_RESULT);
    System.out.println(" The result is : " + result );

    if( code.equals( ResultConstants.VALID_USER ))
    {
        System.out.println( "Authentication succeeded " );
    }
    else if ( code.equals( ResultConstants.NEXT_CODE_REQUIRED))
    {
        System.out.println("Please enter your next code");
        String nextcode = readLine();
        System.out.println("The next code is: " + nextcode);

        //Set the state for next api call
        user.clear();
        user.put( UserConstants.SC_AUTH_STATE,
            result.get( ResultConstants.SC_AUTH_STATE ) );
        putNextCode( user, username, nextcode );
    }
}

```

SecurIdExample continues:

```
        result = api.authenticate(user);
        code = (String)result.get(ResultConstants.AUTHENTICATION_RESULT);
        System.out.println(" The result is: " + result );

        if (code.equals( ResultConstants.VALID_USER ))
            System.out.println("Authentication succeeded on next code ");
        else
            System.out.println("Authentication failed on next code : ");
    }
else if (code.equals(ResultConstants.NEW_PIN_REQUIRED))
    {
        //Set the state for next api call
        System.out.println("Please enter the new PIN : ");
        String pin = readLine();
        System.out.println("The new pin is: " + pin);

        user.clear();
        user.put( UserConstants.SC_AUTH_STATE,
                result.get( ResultConstants.SC_AUTH_STATE ) );
        putPin( user, username, pin );

        result = api.authenticate( user );

        System.out.println( " result is: " + result );

        code = (String)result.get(ResultConstants.AUTHENTICATION_RESULT );
        if (code.equals(ResultConstants.NEW_PIN_ACCEPTED))
            {
                System.out.println("The new pin was accepted, " +
                    "now enter your new pin+passcode.");
                String pass = readLine();

                //Set the state for next api call
                user.clear();
                user.put( UserConstants.SC_AUTH_STATE,
                        result.get( ResultConstants.SC_AUTH_STATE ) );
                putInformation( user, username, pass );

                result = api.authenticate(user);
                code = (String)result.get(
                    ResultConstants.AUTHENTICATION_RESULT);
                System.out.println("code is " + code);
                System.out.println(" The result is " + result );
            }
    }
```

SecurIdExample continues:

```

        else
        {
            System.out.println("Then new PIN was rejected");
        }
    }
    else
    {
        System.out.println( "Authentication failed; reason: "
            + result.get(
                ResultConstants.AUTHENTICATION_RESULT));
    }
}

void putNextCode( Map user, String username, String nextcode )
{
    user.put( UserConstants.SC_USER_ID, username );
    user.put( UserConstants.AUTHENTICATION_TYPE, AuthTypes.SC_SECURID);
    user.put( UserConstants.SC_SECURID_NEXT_CODE, nextcode);
}

void putPin( Map user, String username, String pin )
{
    user.put( UserConstants.SC_USER_ID, username );
    user.put( UserConstants.AUTHENTICATION_TYPE, AuthTypes.SC_SECURID);
    user.put( UserConstants.SC_SECURID_NEW_PIN, pin);
}

void putInformation( Map user, String username, String passcode )
{
    user.put( UserConstants.AUTHENTICATION_TYPE, AuthTypes.SC_SECURID);
    user.put( UserConstants.SC_USER_ID, username );
    user.put( UserConstants.CREDENTIALS, passcode);
    System.out.println(user);
}

static void usage()
{
    System.out.println("The usage is " +
        "java SecurIdExample loop_time ssl_true/false");
}
}

```


7

Administrative and Runtime DCOM API

The RSA ClearTrust® DCOM API allows ASP pages to use the administrative and runtime features of the RSA ClearTrust API. The DCOM API is implemented by accessing the RSA ClearTrust Java API via a bridge layer.

Requirements

Accessing RSA ClearTrust functions from DCOM involves these main components:

- your ASP page containing RSA ClearTrust API calls.
- the Linar J-Integra bridging software version 1.5 (`jintegra.jar` and supporting files from `jintegra_1.5.zip`).
- the RSA ClearTrust API classes, stored as the `ct_dcom.jar`.
- the RSA ClearTrust API Server, which is part of the Entitlements Server.

A typical physical layout is shown in the following diagram. RSA Security recommends installing the J-Integra software and the RSA ClearTrust API classes (the jars) on the machine that will host your ASP pages or other COM programs. The RSA ClearTrust API Server resides on your main RSA ClearTrust server machine with the rest of the RSA ClearTrust Servers.

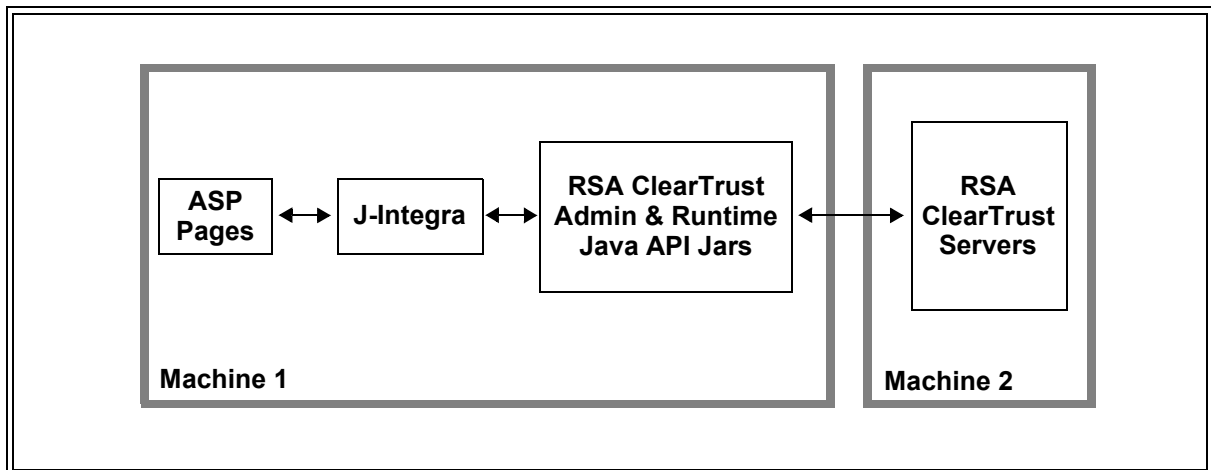


Figure 7.1 Typical installation on two machines

Before beginning the installation steps below, make sure you have the RSA ClearTrust Servers installed on your RSA ClearTrust server machine.

Installing the DCOM API

Perform the following steps on the Web server machine where your ASP pages will run.

1. Make sure the Java Runtime Environment (JRE) version 1.3 is installed on the machine. The JRE software may be downloaded from <http://java.sun.com/j2se/1.3/jre/>
2. Make sure the IIS Web server is installed on the Web server machine in order to host your ASPs. (If you wish to protect this Web server, you may also install the RSA ClearTrust Web server Agent on it, but this is not necessary in order to use the DCOM API.)
3. Install the RSA ClearTrust API classes if you have not already done so as part of an RSA ClearTrust Custom installation (see Chapter 2 of the *RSA ClearTrust Installation and Configuration Guide*):
 - a. Insert the RSA ClearTrust CD, go to the `ct_servers` directory and run the `setup.exe` program.
 - b. Make your selections and click **Next** or **Yes** to advance through the Welcome, Region, License, Destination windows.
 - c. In the Setup Type window, select **Custom**.
 - d. In the next window, click the following check boxes to the ON position: RSA ClearTrust Servers, Documentation, and RSA ClearTrust API (including RSA ClearTrust C API, RSA ClearTrust DCOM API, and RSA ClearTrust Java API).
 - e. Click through the remaining windows to complete the installation. If you have questions, consult the *Installation and Configuration Guide*.

Once the installation is complete, your file system will contain a directory named `<CT_HOME>\api\dcom`. (`<CT_HOME>` is the directory in which you installed RSA ClearTrust.) The `<CT_HOME>\api\dcom` directory contains the RSA ClearTrust Java API packages and the DCOM wrappers for the RSA ClearTrust DCOM API, and it also contains the J-Integra bridging software.

4. Make sure the needed files were installed:


```
<CT_HOME>\api\admin-j\lib\ct_admin_api.jar
<CT_HOME>\api\runtime-j\lib\ct_runtime_api.jar
<CT_HOME>\api\dcom\ct_dcom.jar
<CT_HOME>\api\dcom\jintegra_1.5.zip
<CT_HOME>\api\dcom\runvm.bat
<CT_HOME>\api\dcom\test.asp
```
5. If you will be connecting your RSA ClearTrust API client programs to the RSA ClearTrust Servers over authenticated SSL, you will need the following additional jar files. You will find these in your `<CT_HOME>\lib` directory.
RSA SSL software:
 - `certj.jar`

- jsafe.jar
- jsafeJCE.jar
- rsajsse.jar
- sslj.jar

JCSI Keystore software:

- jcsi_base.jar
- jcsi_provider.jar

Sun security infrastructure:

- jcel_2-do.jar
- jcert.jar
- jnet.jar
- jsse.jar

Make sure these jar file names are included in your CLASSPATH.

6. Install Linar J-Integra by extracting the <CT_HOME>/api/dcom/jintegra_1.5.zip archive into the desired location on your file system. For example, you may wish to install J-Integra in C:\jintegra\. In the rest of these instructions, we will refer to this J-Integra directory as <JINTEGRA_HOME>.
7. Next, edit the runvm.bat script (<CT_HOME>\api\dcom\runvm.bat) to make sure your CLASSPATH is set correctly. Using the CT_ROOT variable to represent the RSA ClearTrust directory, your CLASSPATH should be as follows:

```
%CT_ROOT%\lib\cleartrust.jar;
%CT_ROOT%\api\admin-j\lib\ct_admin_api.jar;
%CT_ROOT%\api\runtime-j\lib\ct_runtime_api.jar;
%JINTEGRA_HOME%\lib\jintegra.jar; %DCOM_API_HOME%\ct_dcom.jar;
%CT_ROOT%\lib\certj.jar; %CT_ROOT%\lib\jcert.jar;
%CT_ROOT%\lib\jcsi_base.jar; %CT_ROOT%\lib\jcsi_provider.jar;
%CT_ROOT%\lib\jsafe.jar; %CT_ROOT%\lib\jsafeJCE.jar;
%CT_ROOT%\lib\jsse.jar; %CT_ROOT%\lib\rsajsse.jar;
%CT_ROOT%\lib\sslj.jar; %CT_ROOT%\lib\jcel_2-do.jar;
%CT_ROOT%\lib\jnet.jar
```



Important: If you copy and paste the above CLASSPATH segment, make sure you remove the spaces from it.

8. Launch the J-Integra Registration GUI by running the `regvm.exe` program. The following screen shot illustrates its use.

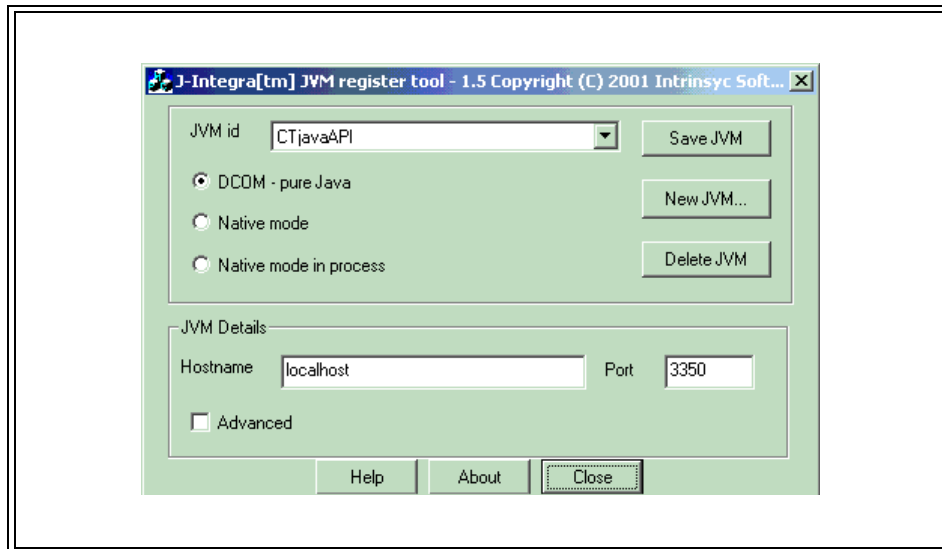


Figure 7.2 The J-Integra Registration GUI

Make the following settings:

- In the **JVM id** field, enter “CTjavaAPI”. This is the name used to refer to the RSA ClearTrust API via J-Integra.
- Select the **DCOM - Pure Java** radio button.
- In the **Hostname** field, enter the hostname of the machine where the J-Integra-wrapped API is available. This should be the machine where your ASP pages will run. (Usually, this is the machine on which you are now installing.) If J-Integra and the RSA ClearTrust APIs are on the same machine, you can simply use “localhost”.
- In the **Port** field, enter the port where the J-Integra-wrapped API is available, RSA Security recommends using “3350”, which is the default port for J-Integra.

Click **Save JVM** to save your settings, and click **Close** to close the window.

If the Registration GUI fails, check your CLASSPATH and try again.

You only need to perform registration once on the machine. If the machine is rebooted, you do not need to register again.

9. Windows 2000 ASP users only: If you are using the RSA ClearTrust DCOM API from Active Server Pages (ASP) and if you are running ASP under Windows 2000 and the ASP virtual directory is set to be Pooled (the default), then edit your COM Authentication Level as shown in the steps that follow.

(Note: If you'd prefer not to change this setting for all pooled applications, configure the virtual directory to be Isolated, and then change the specific entry for that virtual directory that appears under **COM+ Applications**.)

- a. Open the Control Panel (**Start ▶ Settings ▶ Control Panel**)

- b. Open the **Administrative Tools** window.
- c. Open the **Component Services** window.
- d. Open **Computers** ▶ **My Computer** ▶ **COM+ Applications**.
- e. Right click on the **IIS Out-Of-Process Pooled Applications** icon. Select **Properties**.
- f. Click on the **Security** tab.
- g. In the **Authentication Level For Calls** combo box, select **Connect**. (The default is **Packet**, which J-Integra does not currently support.)
- h. Click **OK** to complete your edit.

10. Run the DCOM bridge application using the `runvm.bat` script.

```
<CT_HOME>\api\dcom\runvm.bat
```

The DCOM Bridge must be running at all times.

Your installation is complete. You may now test it. For example, using ASP pages, you can test the API with the following lines. Note that, for this example, we use the hostname `venus.cleartrust.com` to refer to the RSA ClearTrust server machine.

```
Set ServerProxy =  
GetObject("CTjavaAPI:sirrus.api.client.APIServerProxy(string:venus.cleartrust.com,  
int:5601)")  
ServerProxy.connect "admin", "admin1234", "Default Administrative Group", "Default  
Administrative Role"
```

For more a complete explanation of using and testing the DCOM API, continue with the next section.

Using the DCOM API

The RSA ClearTrust DCOM API is a wrapper around the RSA ClearTrust Java API using Linar J-Integra as the bridge. The following diagram shows the relationship between the DCOM API, Java API and J-Integra.

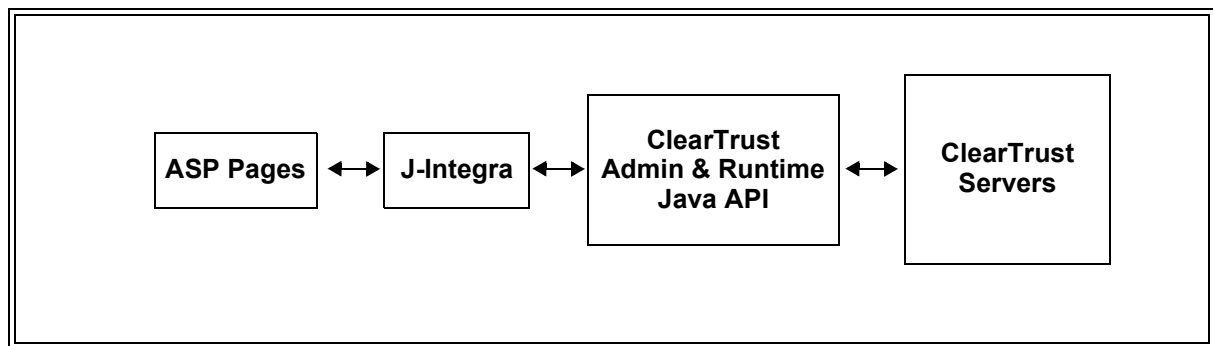


Figure 7.3 RSA ClearTrust's DCOM API structure

The RSA ClearTrust DCOM API can be used in the same way as the RSA ClearTrust Java Administrative and Runtime APIs, with the exception of special syntax you will need to instantiate classes whose constructors take arguments. You will need to use the syntax described below when getting `sirrus.api.client.APIServerProxy` objects for the Administrative API, and when getting `sirrus.runtime.APIFactory` and other objects for the Runtime API.

Instantiating and Connecting

Creating and connecting the `APIServerProxy` Administrative API object provides a simple example of how to connect. For a more complex example including setting up an authenticated SSL connection, see “[DCOM Runtime API example](#)” on page 198.

The `APIServerProxy` constructor takes three arguments:

- The hostname of the machine where the RSA ClearTrust Entitlements Server is running. For example, *venus.cleartrust.com*.
- The port number of the RSA ClearTrust Administrative API. This is usually *5601*.
- A boolean setting indicating whether the Entitlements Server is set up to communicate via SSL. In this example, we will set it to *false*.

Using a VB script, you would call the constructor as follows:

```
serverProxy =getObject("CTjavaAPI:sirrus.api.client.APIServerProxy(string:venus.\cleartrust.com,int:5601,boolean:false)")
```

Specifically, the arguments to the VB call are

- `CTjavaAPI`: This is the registered COM name for the RSA ClearTrust API, as seen through J-Integra. (The installation section above describes how to set up and register this.)
- `sirrus.api.client.APIServerProxy`: The name of the class being instantiated.
- `(string:venus.cleartrust.com,int:5601,boolean:false)`: These are the parameters for the constructor. When the class constructor takes parameters, they are listed in the format, “(type:value,type:value,...,type:value)”. Currently three types are supported: string, int, and Boolean. These are all the types required by the RSA ClearTrust Admin and Runtime APIs. *Note that the separator between parameters is just a comma; no space is allowed after the separator.*

Getting Objects

You can use the `GetObject` call to load most objects.

```
APIFactory = GetObject("CTjavaAPI:sirrus.runtime.APIFactory")
UserConstants = GetObject("CTjavaAPI:sirrus.api.com.UserConstantsClass")
AuthTypes = GetObject("CTjavaAPI:sirrus.api.com.AuthTypesClass")
user = GetObject("CTjavaAPI:java.util.HashMap")
```

Making RSA ClearTrust API Calls

After getting the object, you use it as you would the RSA ClearTrust Java API. For example:

```
serverProxy.connect( "admin", "admin1234", "Default Administrative Group",
"Default Administrative Role" )
    rtAPI = APIFactory.createFromServerDispatcher( dispatcher )
    aResult = rtAPI.authenticate( user )
```

Classes in the `sirrus.api.com` Package

SecurantDCOMFactory

```
sirrus.api.com.SecurantDCOMFactory
public final class SecurantDCOMFactory extends java.lang.Object
```

Description

Java objects run inside a Java Virtual Machine. In order to access them from ASP, they have to be run in a way that is accessible to DCOM. This class creates DCOM-accessible instances of `sirrus.api.client.APIServerProxy` and `sirrus.runtime.RuntimeAPI`.

Methods

```
static sirrus.api.client.APIServerProxy getAdminServerProxy(
    java.lang.String host, int port, boolean ssl)
```

`getAdminServerProxy` returns a reference to an Administrative API Proxy Interface. This method takes three arguments: This constructor takes three arguments: *host* is the String hostname of the machine where the RSA ClearTrust Entitlements Server is running; *port* is the int port number of the RSA ClearTrust Administrative API; and *ssl* is the boolean setting indicating whether the Entitlements Server is set up to communicate via SSL.

```
static sirrus.runtime.RuntimeAPI getRuntimeServerProxy(
    java.lang.String host, int port, boolean useSSL)
```

`getRuntimeServerProxy` returns a reference to a Runtime API Proxy Interface.

AuthTypesClass

```
sirrus.api.com.AuthTypesClass
public class AuthTypesClass extends java.lang.Object implements
sirrus.runtime.AuthTypes
```

Description

This class implements the `AuthTypes`. It allows the DCOM API to access constants in the interface `AuthTypes`.

ResourceConstantsClass

```
sirrus.api.com.ResourceConstantsClass
public class ResourceConstantsClass extends java.lang.Object
implements sirrus.runtime.ResourceConstants
```


Description

This class implements the ResourceConstants. It allows the DCOM API to access constants in the interface ResourceConstants.

ResultConstantsClass

```
sirrus.api.com.ResultConstantsClass  
public class ResultConstantsClass extends java.lang.Object implements  
sirrus.runtime.ResultConstants
```

Description

This class implements the ResultConstants. It allows the DCOM API to access constants in the interface ResultConstants.

UserConstantsClass

```
sirrus.api.com.UserConstantsClass  
public class UserConstantsClass extends java.lang.Object implements  
sirrus.runtime.UserConstants
```

Description

This class implements the UserConstants. It allows the DCOM API to access constants in the interface UserConstants.

UserPropertyTypesClass

```
sirrus.api.com.UserPropertyTypesClass  
public class UserPropertyTypesClass extends java.lang.Object  
implements sirrus.api.client.IUserPropertyTypes
```

Description

This class implements the IUserPropertyTypes. It allows the DCOM API to access constants in the interface IUserPropertyTypes.

DCOM API Example Code

Runtime and Administrative examples are included below. In all of these examples, the registered COM name for the RSA ClearTrust API (as seen through J-Integra) is "CTjavaAPI".

DCOM Runtime API example

The code segment that follows shows how to instantiate a Runtime API client, connect it over authenticated SSL, and authenticate a user. If you wish to connect a non-authenticated Runtime client, you may leave out the keystore-related calls.

In order to use this code as-is, you must define the following variables and set their values as explained below. (Bear in mind that these variable names are just the ones used in this example; they have no particular significance in the RSA ClearTrust DCOM API.)

- `ctServerName` - the string hostname of the Dispatcher machine. For example, "localhost".
- `ctServerPort` - the int port number of the Dispatcher's listener port. Typically, this is 5608.
- `useSSL` - a boolean indicating whether the connection should be SSL-secured. Set it to true if you wish to use SSL.
- `keyStoreType` - the string keystore type name. For example, "PKCS12".
- `keyStoreProvider` - the string keystore provider name. For example, "DSTC_PKCS12".
- `keyStoreFileName` - the string file name of the keystore file. For example, "c:\temp\kca.p12".

DCOM Runtime Example:

```

set UserConstants = GetObject("CTjavaAPI:sirrus.api.com.UserConstantsClass")
set ResultConstants = GetObject("CTjavaAPI:sirrus.api.com.ResultConstantsClass")
set AuthTypes = GetObject("CTjavaAPI:sirrus.api.com.AuthTypesClass")
set TokenKeys = GetObject("CTjavaAPI:sirrus.api.com.TokenKeysClass")

set CredentialConstants
=GetObject("CTjavaAPI:sirrus.api.com.CredentialConstantsClass")

set apiFactory = GetObject("CTjavaAPI:sirrus.runtime.APIFactory")

set keyStoreFactory = GetObject("CTjavaAPI:sirrus.util.crypt.KeyStoreFactory")

set dispatcher = GetObject("CTjavaAPI:sirrus.runtime.ServerDescriptor(string:" &
ctServerName _ & ",int:" & ctServerPort & ",boolean:" & useSSL & ")")

set keyStore =
keyStoreFactory.getKeyStore(keyStoreType,keyStoreProvider,keyStoreFileName )

'create the credential for Authenticated SSL
set credentials = GetObject("CTjavaAPI:java.util.HashMap")

Call credentials.put( CredentialConstants.SC_SSL_KEYSTORE, keyStore )

Call credentials.put( CredentialConstants.SC_TOKENS_ENABLED, "true" )

Call keyStoreFactory.putCharArrayInMap( credentials, _
CredentialConstants.SC_SSL_PRIVATE_KEY_PASSPHRASE,_ "abc123" )

'get the RuntimeAPI object/connection
set rtAPI2 = apiFactory.createFromServerDispatcher( credentials, dispatcher )

set user = GetObject("CTjavaAPI:java.util.HashMap")

user.put UserConstants.SC_USER_ID, CSTR(Request.Form("userid1"))

user.put UserConstants.AUTHENTICATION_TYPE,CSTR(Request.Form("authtype1"))

user.put UserConstants.CREDENTIALS, CSTR(Request.Form("password1"))

set aresult = rtAPI2.authenticate( user )

Dim retcode retcode = aresult.get( ResultConstants.RETURN_CODE )

if retcode = ResultConstants.VALID_USER then Response.Write "Authentication
succeeded"
else Response.Write "Authentication failed; reason: " & _ aresult.get(
ResultConstants.AUTHENTICATION_RESULT ) & ""
end if

```

ASP page, create user

This example shows how to create and save a user with the Administrative API. When you use `GetObject` to instantiate the `APIServerProxy`, you must provide the hostname of the RSA ClearTrust server machine. In this example, we use the hostname `venus.cleartrust.com` to refer to the server machine.

```
<%@ Language=VBScript %>
<HTML>
<HEAD>
</HEAD>
<BODY>

<%
    On Error Resume Next

    ' create our server proxy object
    Set ServerProxy =
GetObject("CTjavaAPI:sirrus.api.client.APIServerProxy(string:venus.cleartrust.com,
int:5601)")

    ' attempt a connect to the api server
    ServerProxy.connect "admin", "admin1234", "Default Administrative Group",
"Default Administrative Role"

    If Err.Description <> "" Then
        Response.write("<font color=red>Error = " & Err.Description & "</font><BR>")
        Err.Description = ""
    End If

    ' Create a new User
    startDate = Date()
    endDate = Date() + Date()

    Set newUser = ServerProxy.createUser( "john" , true, startDate, endDate, _
        "firstname", "lastname1", "john@somewhere.com", "password1",false
    )

    newUser.save()

    if Err.Description <> "" Then
        Response.write("<font color=red>Error = " & Err.Description & "</font><BR>")
    Else
        Response.write("<font color=blue>User Created<BR></font>")
    End if

    ' shut down now
    ServerProxy.disconnect
%>
</form>
</BODY>
</HTML>
```


Example continues:

```
if Err.Description <> "" Then
    Response.write("<font color=red>(Get Users List)Error = " &
Err.Description & "</font><BR>")
    Err.Description = ""
End if
End if

' shut down now
ServerProxy.disconnect
%>
</BODY>
</HTML>
```

8

Web Agent Extension API

Overview

The RSA ClearTrust® Web Server Agents provide the RSA ClearTrust Web Agent Extension API (the “WAX API”), which allows developers to extend and customize the functionality of any RSA ClearTrust Agent. For example, your extensions may perform custom authentication or manipulate dynamic, user-specific content.

Unlike the RSA ClearTrust Administrative API, the WAX API does not enable you to edit entries in the RSA ClearTrust database, rather it allows you to modify the behavior of the RSA ClearTrust Agent during the authentication and authorization processing. An extension you write using this API is called a Web Agent Extension (or “WAX program” for short).

For example, you could extend the functionality of the RSA ClearTrust Web Server Agent in the following ways.

- Create an extension directing the Web server to display, based on the Authorization Server return code, a specific HTML file that corresponds to that return code. (Usually, the return code is a denial of access for a specific reason, so the html page might be one explaining why access was denied.)
- Create an extension providing custom logging.
- Create an extension providing custom authentication of users.
- Create an extension integrating the RSA ClearTrust Agent with proprietary Web Server Agents or third-party Agents.

This chapter provides an overview of the WAX API, including processing flows and logic loops, and shows how to build and integrate your own custom extension—the WAX program that you write—into your Web server environment. It includes the following sections:

- [“Extending the Web Server Agent”](#) on page 204
- [“How an Agent Processes a URI Request”](#) on page 205
- [“Agent Phase Handlers”](#) on page 207
- [“Writing a WAX Program”](#) on page 212
- [“WAX Examples”](#) on page 218
- [“WAX API Reference”](#) on page 227



Note: In this chapter, the RSA ClearTrust Web Server Agent installation directory is referred to as <CT_AGENT_ROOT>. Directories that contain other software are abbreviated in a similar way, such as <IPLANET_SERVER_DIR> and <APACHE_SERVER_DIR>.

Extending the Web Server Agent

RSA ClearTrust WAX programs are implemented using a call-back scheme. Much like the call-back mechanisms in the iPlanet Web server and IIS Web server, a WAX program must register itself to the RSA ClearTrust Agent and define the various routines to call when processing a URL request. A call made using the WAX API is available to the Agent to which it is registered and to the Web server.

Figure 8.1 illustrates the relationships among a WAX program, an RSA ClearTrust Agent, and a Web server, via their respective APIs.

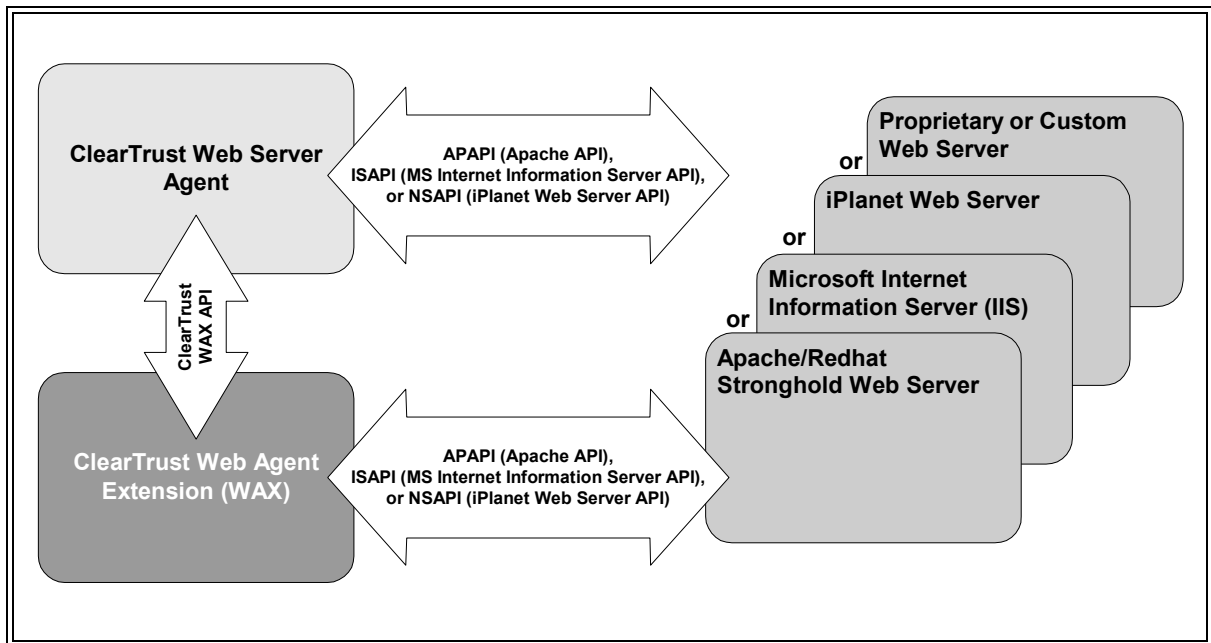


Figure 8.1 RSA ClearTrust Web Agent Extension (WAX) API

As shown in Figure 8.1, RSA ClearTrust Web Server Agents are implemented using the APIs of the respective Web server vendors, and WAX programs also use these server APIs.

Before discussing the specific implementation details, the following section provides a brief overview of the default RSA ClearTrust Web Server Agent multi-phased request handler process.

How an Agent Processes a URI Request

During a URI request, the Web server invokes the RSA ClearTrust Agent to perform authentication and authorization. The RSA ClearTrust Agent processes the request by executing a sequence of *phases*. During each phase, the RSA ClearTrust Agent first invokes a *phase handler* to perform an associated action and then invokes a *status handler* to handle the status from the phase handler. The WAX programs that you write are *phase handlers*.

The status handler determines the next phase to execute or stops the execution, and returns the status to the Web server. Information for each request is passed between the phase handlers and status handler using a hash table. There is a single status handler in the loop, but each phase has its own distinct phase handler (see Figure 8.2).

As the RSA ClearTrust Web Server Agent processes a phase, it first invokes any custom phase handler that is registered. The custom phase handler performs its action and returns a boolean value indicating whether or not it handled the phase.

- If it returns TRUE, the Agent moves on to the next phase (and no additional handlers are invoked for this phase).
- If it returns FALSE, the Agent invokes the next registered handler for this phase, or, if no other handlers are registered, the Agent invokes the default handler for this phase.

You may add as many phase handlers as you need. Using multiple phase handlers is called *WAX chaining*. In a given WAX chain, only the last phase handler returns TRUE; all others return FALSE (regardless of the success or failure of their internal actions), which instructs the Web Agent to invoke the next handler in the current phase.

After the phase handlers have completed, the status handler is invoked to handle the result from the phase handler. The Agent first calls any custom status handler that may exist. Like the phase handler, the custom status handler returns a boolean value indicating whether or not it handled the status.

- If the custom status handler returns TRUE, no other status handlers are invoked. However, the default status handler is still invoked. When the default status handler is invoked, only logging and messaging actions are performed; processing for the URI request ceases.
- If the custom status handler returns FALSE, the next registered handler or default handler, if none, is invoked.

Figure 8.2 shows how the processing logic transfers between default phase handlers and custom phase handlers, and how the status handler receives the return codes that determine the next execution phase.

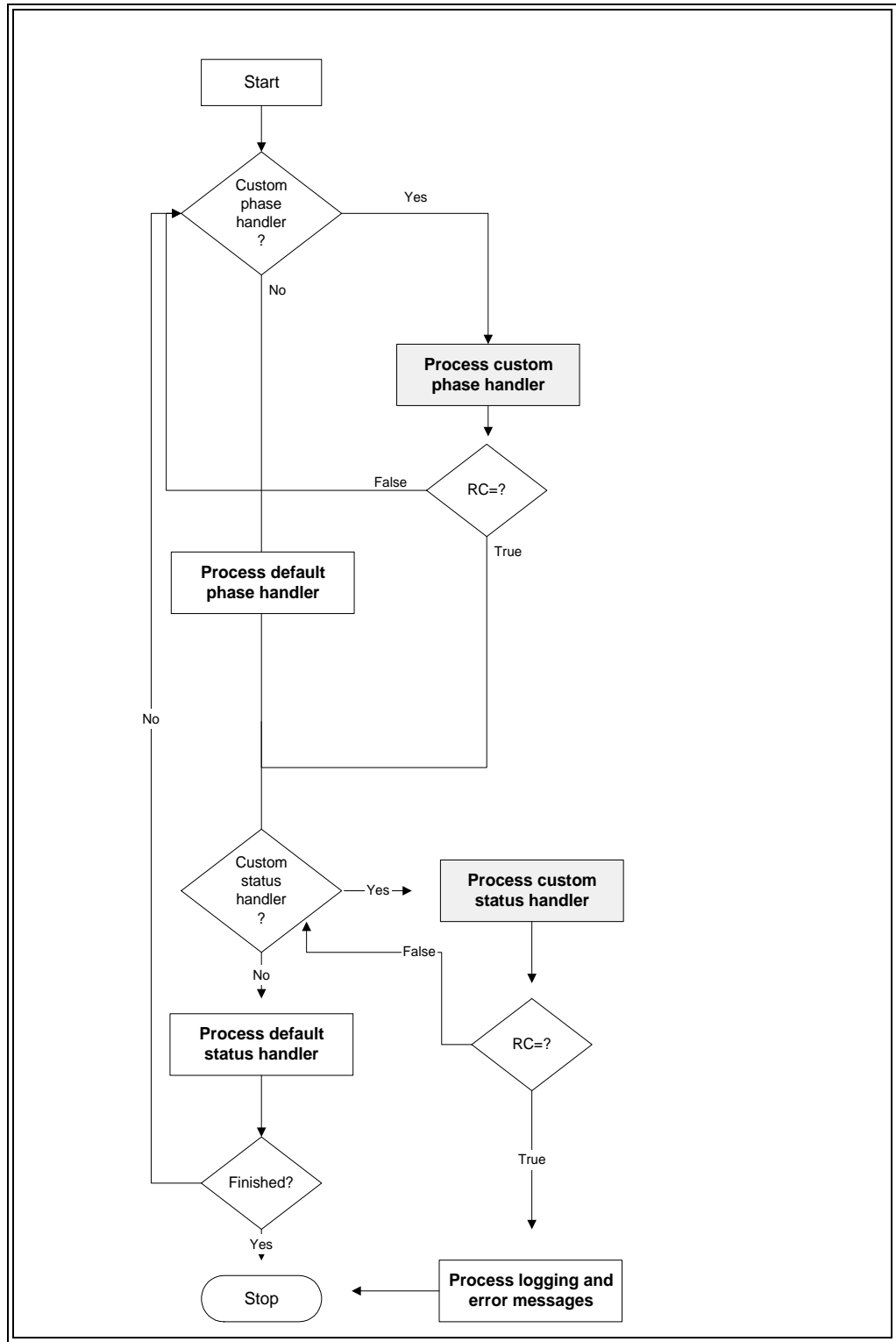


Figure 8.2 RSA ClearTrust Agent Processing Loop

Agent Phase Handlers

During the processing of a URI request, the RSA ClearTrust Web Server Agent executes a sequence of phases to perform authentication, authorization, and single sign-on, ultimately determining the accessibility of the URI. The phase handlers are listed here in the order in which they run:

- Path Check Handler (CT_PATH_CHECK_HANDLER)
- Session Handler (CT_SESSION_HANDLER)
- Pre-Authentication Handler (CT_PREAUTHENTICATION_HANDLER)
- Authentication Handler(s) (CT_AUTHENTICATION_HANDLER)
- Authorization Handler
- Cookie Handler (CT_COOKIE_HANDLER)

In addition, there is a handler called the Status Handler (CT_STATUS_HANDLER) that is driven after each phase.

Path Check Handler

The Path Check Handler determines whether the requested URI is protected. The handler invokes the RSA ClearTrust Authorization Server to perform the path check.

- If the URI isn't protected, the system returns the status code CT_AUTH_URL_UNPROTECTED and the default status handler instructs the Web server to serve the requested URI.
- If the URI is protected, the system returns the status code CT_AUTH_URL_PROTECTED and the default status handler retrieves the list of required and/or allowed authentication types from the Authorization Server. It then inserts this list of authentication types into the request table under the key CT_ALLOWABLE_AUTH_MODES, and calls the next handler.



Note: For each protected resource, an administrator will set the allowed and/or required authentication types in the Agent's <CT_AGENT_ROOT>/conf/webagent.conf file.

Session Handler

The Session Handler determines whether or not the cookie (used for single-sign-on support) has expired.

- If the cookie has expired, the system returns the status code CT_SESSION_EXPIRED and the default status handler sends a WWW-Authenticate response (HTTP 401) to the browser for re-authentication.

- If the cookie has not expired, the system returns the status code `CT_SESSION_ACTIVE` and the next handler is called.

Pre-Authentication Handler

The Pre-Authentication Handler gets the allowed and/or required authentication type(s) required by the URI from the request table (`CT_ALLOWABLE_AUTH_MODES`) and checks to see if the user has already authenticated with all or some of the authentication types.

- If the user has already authenticated with the required authentication type or types, the Pre-Authentication Handler sets a status code of `CT_CHECK_ACCESS_REQUIRED`, causing the Authorization Handler to be invoked next.
- If there are required authentication types for which the user has not authenticated, the Pre-Authentication Handler asks the appropriate Authentication Handler(s) to authenticate the user.

Authentication Handler

The Authentication Handler may be a standard RSA ClearTrust Authentication Handler or a custom one that you have built. In the paragraphs that follow, we show how the Authentication Handler performs RSA ClearTrust BASIC authentication. This behavior provides a good model for building custom authentication handlers.

The Authentication Handler starts with the first, not-yet-satisfied authentication type on the `CT_ALLOWABLE_AUTH_MODES` list. In order to authenticate the user via form-based authentication, the Authentication Handler must get the user's credential (`CT_USER/CT_PASSWORD` or `CT_DN`) from the request table.

- If no user ID or DN is present in the request table, the Authentication Handler sets a status code of `CT_AUTH_BAD_USERNAME`.
- If no password is present in the request table, the Authentication Handler sets a status code of `CT_AUTH_BAD_PASSWORD`.
- If the user's credential exists in the request table (user ID/password or DN), the Authentication Handler attempts to authenticate the user with the appropriate authentication type.

Based on what is returned from the authentication attempt, the Authentication Handler sets the appropriate status code.

- If the authentication is successful, the Authentication Handler sets the authenticated bit (`CT_AUTHENTICATED`) to signify that the user has authenticated successfully with the current authentication type and therefore will not need to authenticate against this mode in the future. A custom Authentication Handler should set the `CT_AUTH_CUSTOM` bit of the `CT_AUTHENTICATED` bitfield to indicate that the custom authentication was successful. Note that the authentication handler should *not* set `CT_AUTH_MODE`; this indicates what authorization type was requested and is set by the Pre-Authentication Handler.

- If the authentication is not successful, the status code is handled as follows:
 - If you are using non-forms-based authentication, the Web server is instructed to return a WWW-Authenticate response (HTTP 401)
 - When forms-based authentication is used, the browser is instead redirected to the configured error message form. You may use the following return codes to choose an appropriate error page to show to the user. (Note that your system will be more secure if you provide users only vague error messages. For example, if a person is trying to break in and sees a “bad password” message, he will know that he has found a valid user name.)
 - CT_AUTH_BAD_USERNAME: The user is not defined in the RSA ClearTrust database.
 - CT_AUTH_BAD_PASSWORD: The password specified does not match the user's password.
 - The Web server is instructed to return a **FORBIDDEN** response (HTTP 403) or is redirected to a custom error page for the following return codes:
 - CT_AUTH_EXPIRED_ACCOUNT: The account has expired.
 - CT_AUTH_INACTIVE_ACCOUNT: The account has not started yet.
 - CT_AUTH_PASSWORD_EXPIRED: The user's password has expired; it must be reset.
 - CT_AUTH_PASSWORD_EXPIRED_FORCED: The user's password has expired via administrative action; it must be reset.
 - CT_AUTH_PASSWORD_EXPIRED_NEW_USER: The user is logging in for the first time; the password must be reset.
 - CT_AUTH_USER_LOCKED_OUT: An administrator has explicitly locked out the user.

Authorization Handler

The Authorization Handler determines whether or not a user has access to the requested URI.

The Authorization Handler invokes the Authorization Server to perform the authorization checking and sets the status code with the returned value.

- If the Authorization Server returns a CT_AUTH_URL_ACCESS_DENIED, the Web server is instructed to return a FORBIDDEN response (HTTP 403) or is redirected to a custom error page.
- If the Authorization Server returns a CT_AUTH_URL_ACCESS_ALLOWED, the Authorization Handler sets the status code to CT_CREATE_COOKIE to instruct the Status Handler to invoke the Cookie Handler.

Cookie Handler

The Cookie Handler creates a cookie to send back to the user with a successful request for a protected URI and adds the following data from the request table:

- If the Cookie Handler successfully creates a cookie, it sets the status to `CT_AUTH_URL_ACCESS_ALLOWED` and the default status handler directs the Web server to serve up the requested URI.
- If there was an error creating the cookie, the Cookie Handler sets a status of `CT_COOKIE_ERROR` and instructs the Web server to return a `SERVER_ERROR` (HTTP 500) to the browser.

The RSA ClearTrust Agent provides users with a 2 Kb data buffer within the cookie that can be used for personalization or custom development. You can use this data buffer to provide additional functionality to an RSA ClearTrust cookie. For example, you may want to create a WAX program to add an e-mail address or other user attributes to a cookie. Another option is to utilize the cookie for user management functions like encryption. Refer to the section titled “*Request Data*” for more information on this data buffer. Figure 8.3 illustrates how the phase handlers process a URI request.

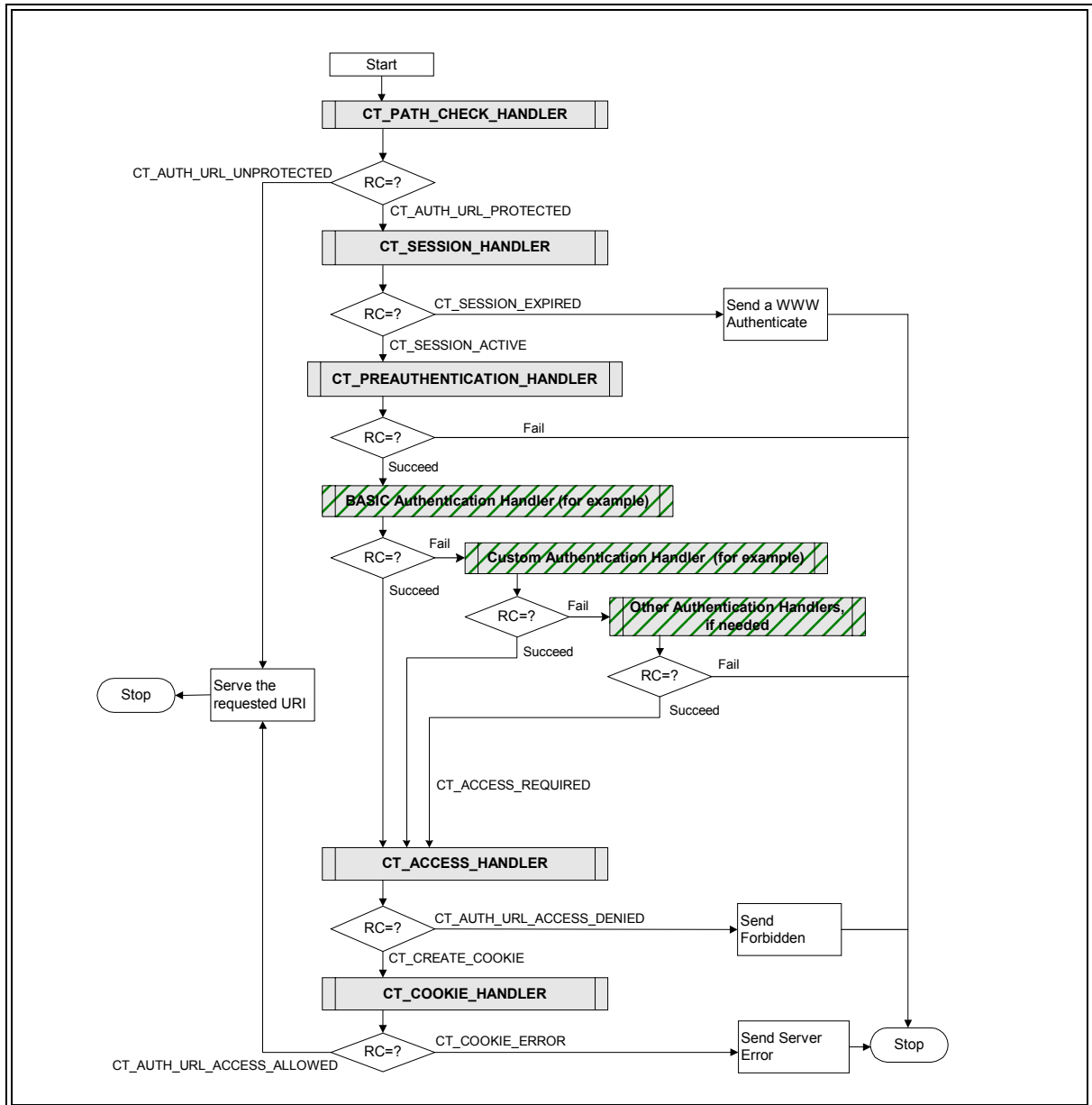


Figure 8.3 Phase Processing a URI Request

Writing a WAX Program

If you wish to modify the behavior of the RSA ClearTrust Web Server Agent (for example, to add logging or to implement a custom authentication mechanism), you can extend the functionality of the RSA ClearTrust Web Server Agent by using the WAX API to write your own phase handler, which can then be called during URI request processing. A custom phase handler is called a WAX program. This section shows how to build a WAX and integrate it into the RSA ClearTrust Web Server Agent processing loop.

Overview

Assuming the RSA ClearTrust Web Server Agent is installed and integrated with your Web Server, adding a WAX program involves the following steps:

1. Write a WAX program using the WAX API. In the `ct_wax_init()` methods of your WAX program, link each of your WAX methods with one or more of the Agent's standard phase handlers.
2. Compile it.
3. Register your WAX program with the Agent by adding its name to the `cleartrust.agent.wax` parameter in the `webagent.conf` file.
4. If your WAX program provides user authentication, then you must edit the `cleartrust.agent.auth_resource_list` in the `webagent.conf` file to specify an authentication type of "CUSTOM" for those resources that require authentication to be performed by your WAX method.

The sections that follow explain these steps in detail and provide the background information you will need to build your WAX program.

WAX API Headers

The WAX API is a C API that is installed as part of the RSA ClearTrust Web Server Agent installation. You will find the header files in the `<CT_AGENT_ROOT>/<Web server type>/include` subdirectory, where "`<Web server type>`" is a directory name that corresponds to your Web server vendor, such as "IIS Agent" or "iPlanet Agent." Note that the `<CT_AGENT_ROOT>` directory is often installed in a separate location from the main `<CT_HOME>` Server installation directory.

The WAX API consists of the following include files:

- `ct_auth_result.h`
- `ct_external.h`
- `ct_function_table.h`
- `ct_memory.h`

- `ct_request_data.h`
- `ct_table.h`

WAX API Libraries

The WAX API works with the library file that is specific to the Web Server Agent that you are extending. Depending on your type of Web server and operating system, you will find your Agent libraries in one of the following locations:

- Microsoft IIS on Windows NT/2000
`<CT_AGENT_ROOT>\IIS Agent\lib\ct_iis50_agent.lib`
- iPlanet Web server on Windows NT/2000
`<CT_AGENT_ROOT>\iPlanet Server Agent\lib\ct_iplanet_agent.lib`
- iPlanet Web server on Solaris
`<CT_AGENT_ROOT>/agent/iplanet/lib/libct_iplanet_agent.so`
- Apache/Redhat Stronghold 3 Web server on Solaris
`<CT_AGENT_ROOT>/agent/apache/lib/libct_apache_agent_apache_ssl.so`
or
`<CT_AGENT_ROOT>/agent/apache/lib/libct_apache_agent_mod_ssl.so`
or
`<CT_AGENT_ROOT>/agent/apache/lib/libct_apache_agent_non_ssl.so`

For more information, see the “Compiling and Linking” sections later in this chapter.

Registering a WAX Program

Register your WAX program with the RSA ClearTrust installation by adding its full path name (the name of the `.dll` or `.so` file) to the `cleartrust.agent.wax` parameter in the Agent's configuration file (`<CT_AGENT_ROOT>/.../conf/webagent.conf`).

Writing a WAX Method

Inside your WAX method, you may implement any functionality you like. Note that your WAX program can read the Agent's configuration parameters (as loaded from the Agent's `webagent.conf` file), which can include custom parameters you have added to the `webagent.conf` file. For details, see [“Loading Parameter Settings”](#) on page 233.

A WAX method must return `TRUE` if it handled its Web Agent phase (thus skipping the default handler for that phase), or `FALSE` if it did not handle the phase (thus allowing the default handler to run as soon as the WAX method returns). In other words, if your WAX method *augments* the handler with which it is associated, then it should return `FALSE`. If your WAX method *replaces* (i.e. *overrides*) the handler with which it is associated, then it should return `TRUE`.

If your WAX method is an authentication method (registered with the `CT_AUTHENTICATION_HANDLER`), it must set the `CT_AUTH_CUSTOM` bit of the `CT_AUTHENTICATED` bitfield if the custom authentication is successful. See [“Authentication Handler”](#) on page 208.

Registering a WAX Method

The method(s) of your WAX program must be associated with one or more of the standard RSA ClearTrust phase handlers, so that the WAX methods will be invoked when those standard phase handlers are called. The phase handlers and the status handler are defined in a function table (a hash table consisting of handler keys and their associated function pointers). The keys define the various phase handlers that comprise the Web Server Agent.

To customize the action that results from a phase handler or to alter the flow of URI request processing, you must register your WAX methods with phase handlers in this function table. You do this using the `ct_table_put` function call. Typically, registration of handlers is performed during initialization of the WAX program, inside the WAX's `ct_wax_init` method, as shown below. (In some cases, registration may be done in a platform/Web server-specific method.) Note also that your `ct_wax_init()` method should return 1 to indicate success.

```
int ct_wax_init(ct_table_ptr ct_func_table, ct_table_ptr config)
{
    ct_table_put(ct_func_table, CT_AUTHENTICATION_HANDLER, my_custom_auth);
    return 1;
}
```

When you call `ct_table_put`, the arguments are, in order:

- `ct_func_table`. This is the function table passed in by the Agent when it called `ct_wax_init()`.
- the handler key of the phase handler (see list below)
- the name of your WAX method to be associated with the phase handler

The list of handler keys and the function table structure are found in the `ct_function_table.h` header file, which is contained in the Agent's installation directory. The handler keys are:

- `CT_STATUS_HANDLER`
- `CT_SESSION_HANDLER`
- `CT_PATH_CHECK_HANDLER`
- `CT_PREAUTHENTICATION_HANDLER`
- `CT_AUTHENTICATION_HANDLER`
- `CT_ACCESS_HANDLER`
- `CT_COOKIE_HANDLER`

See [“Agent Phase Handlers”](#) on page 207 for a description of each handler.

Keep in mind that your configuration can include multiple WAX methods. Multiple WAX methods may be associated with a single phase handler, and more than one phase handler may have WAX methods associated with it.



Warning: If you use the `ct_print` method in a WAX program, you must format the statement correctly, otherwise it could cause your Web server to crash. In particular, make sure you do not pass in more format specifiers (for example, "%s" conversion flags) than there are parameters to fill them.

Invoking a WAX Authentication Method

If your WAX method performs *authentication*, it will only be called when a user attempts to load a resource that is protected by the CUSTOM (SC_AUTH_TYPE_CUSTOM) authentication type. In contrast, if your WAX method is a *non-authentication* method (such as a logging method), it will be called every time its associated phase handler(s) is/are called.



Note: Note that a WAX authentication method must be registered with the `CT_AUTHENTICATION_HANDLER`; custom authentication cannot be associated with any other handler

To use your WAX authentication method, edit the `webagent.conf` file to specify an authentication type of "CUSTOM" for all resources that require authentication to be performed by your WAX method. To do this, add the names of these resources or directories of resources to the `cleartrust.agent.auth_resource_list` with their authentication type set to CUSTOM. Alternatively, you may set your `cleartrust.agent.default_auth_mode` to CUSTOM so that all resources will use your custom WAX authentication unless specified otherwise. See the "Authentication Parameters" section of Appendix A in the *RSA ClearTrust Installation and Configuration Guide*.

Compiling and Linking a WAX Program

The following sections provide guidelines for compiling and linking with the WAX API on the following platforms:

- Microsoft IIS
- iPlanet Web server on Windows NT
- iPlanet Web server on UNIX
- Apache Web server on UNIX

Compiling and Linking for Microsoft IIS

This section tells you how to compile and link with the WAX API for Microsoft IIS. You must use Microsoft Visual C++ 6.0 to compile WAX programs; the Agent itself is now compiled with 6.0, so WAX programs can no longer be compiled with Visual C++ 5.0.

Compiling

Compile Options: WIN32, _DEBUG, _WINDOWS, MSIIS, WINDOWS

Additional Include Directories: <CT_AGENT_ROOT>\IIS Agent\include

Linking

Additional Libraries: ct_iis50_agent.lib

Additional Library Path: <CT_AGENT_ROOT>\IIS Agent\lib

Compiling and Linking for iPlanet Web Server on Windows NT

This section tells you how to compile and link with the WAX API for iPlanet Web server on Windows NT. You must use Microsoft Visual C++ 6.0 to compile WAX programs; the Agent itself is now compiled with 6.0, so WAX programs can no longer be compiled with Visual C++ 5.0.

Compiling

Compile options: WIN32, _DEBUG, _WINDOWS, NETSCAPE, WINDOWS, XP_WIN32

Additional include directories:

<CT_AGENT_ROOT>\iPlanet Server Agent\include

and

<IPLANET_SERVER_DIR>\include

Linking

Additional library: ct_iplanet_agent.lib

Additional library path: <CT_AGENT_ROOT>\iPlanet Server Agent\lib

Compiling and Linking for iPlanet Web Server on UNIX

This section tells you how to compile and link with the WAX API for iPlanet Web server on UNIX.

Compiling

Compile options: -DNETSCAPE -DFILE_UNIX -DXP_UNIX

Additional include directories: <CT_AGENT_ROOT>/agent/ipplanet/include

and <IPLANET_SERVER_DIR>/include

Linking

Additional libraries: `libct_ipplanet_agent.so`

Additional library path: `<CT_AGENT_ROOT>/agent/ipplanet/lib`

Recommended link option: `-Bsymbolic`

The `-Bsymbolic` option forces the WAX to resolve the symbols against the API rather than the Agent. If you do not use it, you may encounter name conflicts with methods in the Agent. If you are using `gcc` to compile, the syntax is different, as shown below.

Recommended link options, if you are using gcc: `-Wl,Bsymbolic`

Compiling and Linking for Apache/Redhat Web Server on UNIX

This section tells you how to compile and link with the WAX API for an Apache Web server or Redhat Stronghold 3 Web server on UNIX.

Compiling

Compile options: `-DAPACHE -DFILE_UNIX -DXP_UNIX`

Additional include directories: `<CT_AGENT_ROOT>/agent/apache/include`

and `<APACHE_SERVER_DIR>/src/include`

and `<APACHE_SERVER_DIR>/src/os/unix`

Linking

Additional libraries:

`<CT_AGENT_ROOT>/agent/apache/lib/libct_apache_agent_apache_ssl.so`

or

`<CT_AGENT_ROOT>/agent/apache/lib/libct_apache_agent_mod_ssl.so`

or

`<CT_AGENT_ROOT>/agent/apache/lib/libct_apache_agent_non_ssl.so`

If your WAX application happens to use the RSA ClearTrust Runtime API, then you must also include that library, `<CT_HOME>/api/runtime-c/lib/ct_runtime_api.so`

Recommended link option: `-Bsymbolic`

The `-Bsymbolic` option forces the WAX to resolve the symbols against the API rather than the Agent. If you do not use it, you may encounter name conflicts with methods in the Agent. If you are using `gcc` to compile, the syntax is different, as shown below.

Recommended link options, if you are using gcc: `-Wl,Bsymbolic`

WAX Examples

The following examples show how to write and register a WAX program:

- [“Cookie Data Example”](#), below
- [“Custom Authentication Example”](#) on page 221
- [“Custom Error Pages Example”](#) on page 224

Cookie Data Example

The following example, `wax.c`, is a WAX example that shows how you can add data to the RSA ClearTrust cookie. Below are the steps that a WAX goes through when it registers itself and when it runs (when the Agent invokes it).

1. Upon start-up, the Agent checks the `cleartrust.agent.wax` parameter in its `webagent.conf` file and loads all the WAX programs listed there. In this example, you might set the parameter similar to the following:

```
cleartrust.agent.wax=D:\\wax_programs\\lib\\wax.so
```

2. The Agent calls the `ct_wax_init` method in your WAX program. By calling this method, the Agent associates (in the function table, `ct_func_table`) the WAX's method(s) with the Agent's phase handler(s). When those phase handlers run, they will automatically call the associated methods in the WAX.

In this example, there is just one WAX method, `my_cookie_phase_handler()`, and it is associated with the authentication handler (`CT_COOKIE_HANDLER`).

3. When a WAX method runs, it sets the status and returns `TRUE` if the it handled this phase (thus skipping the default handler), or it returns `FALSE` if it did not handle the phase. See [“Writing a WAX Method”](#) on page 213 for details.

In this example, the WAX merely augments the `CT_COOKIE_HANDLER`, so it returns `FALSE` indicating that the cookie handler should still run.

wax.c Example

```
/*
 * wax.c
 *
 * This example, wax.c, is a Web Agent Extension (WAX) that
 * shows how you can insert data into the RSA ClearTrust cookie.
 *
 * Last updated February 10, 2002.
 */

// The standard includes
#include <stdio.h>
#include <string.h>

// Only include windows.h on Windows
#ifdef _WINDOWS
    #include <windows.h>
#endif

// ClearTrust includes
#include "ct_function_table.h"
#include "ct_request_data.h"
#include "ct_external.h"

// Internal macros
#define SUCCESS          1
#define FAILURE          0

#define EMAIL            "spasam@rsasecurity.com"
#define EMAIL_LENGTH    strlen(EMAIL)

// Prototypes declaration

/**
 * This function is the initial interface between ClearTrust Agent
 * and the Web Agent eXtension (WAX).
 */
CT_EXTERNAL int ct_wax_init (ct_table_ptr, ct_table_ptr);

/**
 * The "Cookie phase handler". This function is invoked by ClearTrust
 * Agent before the ClearTrust Cookie Phase is handled.
 */
CT_EXTERNAL int my_cookie_phase_handler (const ct_server_parms *,
                                         ct_table_ptr);
```

wax.c Example Continues

```

/**
 * This function is the initial interface between ClearTrust Agent
 * and the Web Agent eXtension (WAX). The function registers the custom
 * phase handlers. In this case, we are registering a custom cookie
 * phase handler.
 */
CT_EXTERNAL int ct_wax_init (ct_table_ptr ct_func_table,
                             ct_table_ptr configuration)
{
    ct_print ("ct_wax_init is invoked ...\n");

    ct_table_put (ct_func_table,
                  CT_COOKIE_HANDLER,
                  my_cookie_phase_handler);

    return SUCCESS;
} // End of ct_wax_init

/**
 * The "Cookie phase handler". This function is invoked by ClearTrust
 * Agent before the ClearTrust Cookie Phase is handled.
 */
CT_EXTERNAL int my_cookie_phase_handler (const ct_server_parms * server_parms,
                                          ct_table_ptr ct_request_table)
{
    ct_print ("my_cookie_phase_handler is invoked ...\n");

    ct_table_put (ct_request_table, CT_USER_DATA, (void *) EMAIL);
    ct_table_put (ct_request_table, CT_USER_DATA_LEN, (void *) EMAIL_LENGTH);

    return FAILURE;
} // End of my_cookie_phase_handler

```


Custom Authentication Example

The following example, `nt_auth.c`, is a WAX example that shows how you can add a custom authentication routine. This example replaces the RSA ClearTrust authentication with native NT authentication. This example uses NT authentication for purposes of demonstration. You would not actually build such a WAX program, since NT authentication is a standard feature of the RSA ClearTrust system.

Example Overview

During Web server initialization, the `nt_auth.c` WAX registers its authentication handler in the function table. When the authentication handler is driven, the `nt_auth.c` WAX checks whether or not a user name and password have been set. If they have been set, the WAX performs NT native authentication, sets the status, and returns TRUE, which indicates that it has succeeded in handling the authentication phase.

How the WAX Registers and Runs

Below are the steps that a WAX goes through when it registers itself and when it runs (when the Agent invokes it).

1. Upon start-up, the Agent checks the `cleartrust.agent.wax` parameter in its `webagent.conf` file and loads all the WAX programs listed there. In this example, you might set the parameter similar to the following:

```
cleartrust.agent.wax=D:\\wax_programs\\lib\\nt_auth.so
```

2. The Agent calls the `ct_wax_init` method in your WAX program. By calling this method, the Agent associates (in the function table, `ct_func_table`) the WAX's method(s) with the Agent's phase handler(s). In this case, the method is associated with the Authentication Handler.

When the Authentication Handler runs, it calls the associated WAX method *if and only if* the requested resource (URI) is protected by the CUSTOM authentication type. See [“Invoking a WAX Authentication Method”](#) on page 215.

In this example, there is just one WAX method, `nt_authenticate`, and it is associated with the authentication handler (`CT_AUTHENTICATION_HANDLER`).

3. When a WAX method runs, it sets the status and returns TRUE if the it handled this phase (thus skipping the default handler), or it returns FALSE if it did not handle the phase. See [“Writing a WAX Method”](#) on page 213 for details.

In this example, the WAX will handle the authentication phase if a user name and password are provided.

Tips for Compiling and Running

This sample works with IIS Web servers and iPlanet Web servers on NT only. To compile for IIS, set the MSIIS compiler directive, for iPlanet Web server set the NETSCAPE compiler directive. See [“Compiling and Linking a WAX Program”](#) on page 215 for details.

In order for this example to work, the user ID names of the registered users in the RSA ClearTrust system must be identical to their NT user ID names.

NT authentication example:

```

/*
 * nt_auth.c
 *
 * Last updated February 19, 2002.
 */

#include <stdio.h>

// Windows header files
#include <windows.h>
#include <winnt.h>

// ClearTrust header files
#include "ct_auth_result.h"
#include "ct_function_table.h"
#include "ct_request_data.h"
#include "ct_external.h"

// Prototype for the NT authentication
CT_EXTERNAL int nt_authenticate(const ct_server_parms *server_parms,
                                ct_table_ptr ct_req_table);

/**
 * Initialization method for Web Agent Extension.
 */
CT_EXTERNAL int ct_wax_init(ct_table_ptr ct_func_table,
                            ct_table_ptr conf)
{
    ct_table_put(ct_func_table,
                CT_AUTHENTICATION_HANDLER,
                nt_authenticate);

    return 1;
}

/*
 * Routine to perform NT authentication. It first calls the
 * LogonUser API to perform NT authentication, then sets the
 * appropriate ClearTrust status. This routine only returns TRUE
 * if the user and password was set. If the user and password
 * isn't set, it lets the default authentication execute which
 * will end up prompting the user for the user and password.
 * Once the user and password is set, the NT authentication WAX
 * makes the NT API call authenticating the user and then

```

NT authentication example continues:

```
* returns a TRUE directing ClearTrust not to perform default
* authentication.
*/
CT_EXTERNAL int nt_authenticate(const ct_server_parms *server_parms,
                               ct_table_ptr ct_req_table)
{
    BOOL bHandled = FALSE;           // If no user or pw, then don't handle
    BOOL bIsAuthenticated = FALSE;
    HANDLE hToken = 0;
    LPTSTR lpszUser = ct_table_find(ct_req_table, CT_PLUGIN_USER);
    LPTSTR lpszPassword = ct_table_find(ct_req_table,
                                       CT_PASSWORD);

    // If the username and password are supplied, we'll handle
    // authentication.
    if (lpszUser != NULL && lpszPassword != NULL)
    {
        // Perform NT authentication. We specify a NULL domain
        // name so the User will be searched through out all the
        // PDCs. Also, we call the NT method LogonUser() with
        // LOGON32_LOGON_NETWORK logon type because we are just
        // authenticating the user, not creating a process
        // under the User's account.
        bIsAuthenticated = LogonUser(lpszUser,
                                     NULL,
                                     lpszPassword,
                                     LOGON32_LOGON_NETWORK,
                                     LOGON32_PROVIDER_DEFAULT,
                                     &hToken);

        // If isAuthenticated isn't 0, then the user is authenticated
        if (bIsAuthenticated)
        {
            ct_table_put(ct_req_table,
                        CT_AUTHENTICATED,
                        (void *) CT_AUTH_CUSTOM);

            // Force access checking
            SET_STATUS(ct_req_table, CT_CHECK_ACCESS_REQUIRED);
        }
    }
}
```

NT authentication example continues:

```

else
{
    DWORD dwError = GetLastError();
    // Set the appropriate ClearTrust Error code
    switch(dwError)
    {
        case ERROR_LOGON_FAILURE:
            SET_STATUS(ct_req_table, CT_AUTH_BAD_USERNAME);
            break;

        default:
            SET_STATUS(ct_req_table, CT_AUTH_UNKNOWN_ERROR);
            break;
    }
}
bHandled = TRUE; // This indicates that we have handled the
                // authentication stage and the Agent can
                // proceed directly to the status handler.
}
return bHandled;
}

```

Custom Error Pages Example

You can use the WAX API to return a custom page when a user is denied access to an RSA ClearTrust-protected resource.

The following WAX program, `redirect.c`, shows how to replace the requested URI with a new URI to display an error code that RSA ClearTrust returns.

During Web server initialization, the WAX programs registers its status handler in the function table. When the status handler is driven, it checks the current status. If it is an error, it replaces the requested URI with a corresponding custom error page and sets the status to `CT_AUTH_URL_ACCESS_ALLOWED` (that is, access is allowed *for the error page only*). This forces the RSA ClearTrust Agent to serve up the new URI.

This example demonstrates how you can extend the functionality of the RSA ClearTrust Agent. This sample works with IIS, iPlanet Web server on NT, and iPlanet Web server on UNIX. To compile for IIS, set the `MSIIS` compiler directive, for iPlanet Web server set the `NETSCAPE` compiler directive.

Custom error pages example:

```
/*
 * redirect.c
 *
 * Last updated February 19, 2002.
 */

#include <stdio.h>

// Windows header files
#include <windows.h>
#include <winnt.h>

// ClearTrust header files
#include "ct_auth_result.h"
#include "ct_function_table.h"
#include "ct_request_data.h"
#include "ct_external.h"

// Prototype for status handler
CT_EXTERNAL int handle_status(const ct_server_parms *server_parms, ct_table_ptr
ct_req_table);

// Module Definitions for the customer error pages
#define BAD_USER_PAGE "/cleartrust/bad_user.html"
#define INVALID_ACCOUNT_PAGE "/cleartrust/invalid_account.html"
#define USER_FORBIDDEN_PAGE "/cleartrust/forbidden_user.html"

/**
 * Initialization method for Web Agent Extension.
 */
CT_EXTERNAL int ct_wax_init(ct_table_ptr ct_func_table, ct_table_ptr conf)
{
    ct_table_put(ct_func_table, CT_STATUS_HANDLER, handle_status);
    return 1;
}

/**
 * Status Handler. Checks for the error codes which we want to return a
 * custom error page. If we are returning a custom error page, then
 * set the return code to TRUE indicating that we handled the status.
 */
```

Custom error pages example continues:

```

* Also, set Status to CT_AUTH_URL_ACCESS_ALLOWED forcing the serving
* of the custom error page
*
*/
CT_EXTERNAL int handle_status(const ct_server_parms *server_parms,
                             ct_table_ptr ct_req_table)
{
    BOOL bHandled = FALSE;

    // Switch off the current status
    switch(GET_STATUS(ct_req_table))
    {

        // If we have a bad user name, then we have an un-registered
        // user. Serve up registration page.
        case CT_AUTH_BAD_USERNAME:
            // Since BAD_USERNAME is returned if no User Name has been
            // supplied, we only want to redirect if one is supplied
            if (ct_table_find(ct_req_table, CT_PLUGIN_USER) != NULL)
            {
                ct_table_put(ct_req_table, CT_URI, BAD_USER_PAGE);
                SET_STATUS(ct_req_table, CT_AUTH_URL_ACCESS_ALLOWED);
                bHandled = TRUE;
            }
            break;

        // If we have an expired or inactive account, then we need
        // to re-register the user
        case CT_AUTH_EXPIRED_ACCOUNT:
        case CT_AUTH_INACTIVE_ACCOUNT:
            ct_table_put(ct_req_table, CT_URI, INVALID_ACCOUNT_PAGE);
            SET_STATUS(ct_req_table, CT_AUTH_URL_ACCESS_ALLOWED);
            bHandled = TRUE;
            break;

        // If the user is denied, serve up a custom error page.
        case CT_AUTH_URL_ACCESS_DENIED:
            ct_table_put(ct_req_table, CT_URI, USER_FORBIDDEN_PAGE);
            SET_STATUS(ct_req_table, CT_AUTH_URL_ACCESS_ALLOWED);
            bHandled = TRUE;
            break;

        default:
            break;

    }
    return bHandled;
}

```

WAX API Reference

The `ct_wax_init` Initialization Method

WAX programs loaded via the `cleartrust.agent.wax` directive must implement the `ct_wax_init()` method. This method is the WAX initialization entry point and is called by the Web Agent upon loading the WAX program. The method is called once for each registered WAX during startup of the Web server, and may be used to register phase handlers and initialize global variables.



Warning: The `ct_wax_init()` method replaces `ct_function_init()` as the WAX program initialization method. The `ct_function_init()` method is deprecated. WAX programs should be modified to use `ct_wax_init()` instead of the old method. Note that method's signature has changed. This method, as opposed to the deprecated `ct_extension_init()`, provides access to the table of configuration parameters for the virtual host. WAX programs may now include additional extension-specific configuration parameters in the RSA ClearTrust `webagent.conf` file, and the normal scoping rules for configuration parameters will apply.

Signature

```
int ct_wax_init(ct_table_ptr func_table, ct_table_ptr config)
```

Parameters

- `func_table` is a pointer to the function table for registering handlers.
- `config` is a pointer to a table containing the Agent's configuration parameters, expressed as name/value pairs. These are the parameter settings loaded from the Agent's `webagent.conf` file. For details, see [“Loading Parameter Settings”](#) on page 233.

Return value

Returns a non-zero value on success, 0 (zero) otherwise.



Note: Because the function table is explicitly provided, there is no need for WAX programs to call `ct_get_function_table()`. `ct_get_function_table()` is now deprecated as it provides indeterminate results in virtual server environments.

Usage

Define your `ct_wax_init` method to call `ct_table_put` to add rows associating each WAX method with a phase handler. See [“Registering a WAX Method”](#) on page 214 for an example.

ct_extension_init

ct_extension_init is deprecated. Use ct_wax_init instead, as explained above.

Hash Table Functions

The RSA ClearTrust Agent uses hash tables for the function table (the list of phase handlers and, optionally, their WAX associations), for the configuration parameters (as loaded from webagent.conf) and for the request data. The hash table functions are located in the ct_table.h header file. The following table lists the hash table functions.

Table 8.1 Hash Table Functions

Code	Function
void ct_table_put(ct_table_ptr table, const char* key, const void* value)	Adds or replaces a value specified by the key. Note: the value is NOT copied, rather the pointer to the value is stored.
void* ct_table_find(ct_table_ptr table, const char* key)	Returns the value pointer for the key. If the key doesn't exist, a NULL is returned.
void ct_table_remove(ct_table_ptr table, const char* key)	Removes the value pointer for the key.
void ct_table_replace(ct_table_ptr table, const char* key, const void* value);	Replaces the current value in the table with the new value.

Memory Management

You should use the RSA ClearTrust ct_malloc() and ct_free() methods to allocate or free memory on your RSA ClearTrust-protected Web servers, rather than using the standard C methods. Consult the comments in ct_function_table.h and in ct_memory.h for complete information.

Table 8.2 Memory Management Functions

Code	Function
void* ct_req_alloc(const ct_server_parms* server_parms, size_t size);	Allocates memory which will be freed automatically at the end of the request. Where possible, RSA Security recommends that you use ct_req_malloc() rather than ct_malloc().
char* ct_req_strdup(const ct_server_parms* server_parms, const char* str);	Copies a string using memory which will be freed automatically at the end of the request.

Table 8.2 Memory Management Functions

Code	Function
<code>void* ct_malloc(size_t size);</code>	ClearTrust replacement for the standard C <code>malloc()</code> call. Where possible, RSA Security recommends that you use <code>ct_req_malloc()</code> (see above) rather than <code>ct_malloc()</code> . If you use <code>ct_malloc()</code> , you must call <code>ct_free()</code> to free the memory before your WAX exits.
<code>void* ct_realloc(void* buf, size_text_size);</code>	ClearTrust replacement for the standard C <code>realloc()</code> call.
<code>void ct_free(void* ptr);</code>	ClearTrust replacement for the standard C <code>free()</code> call. Frees memory that was allocated using <code>ct_malloc()</code> . Not needed if you use <code>ct_req_malloc()</code> .
<code>char* ct_strdup(const char* str);</code>	ClearTrust replacement for the standard C <code>strdup()</code> call.

Printing Status and Debug Information

You should use the RSA ClearTrust `ct_print()` method to print status and debug information.



Warning: If you use the `ct_print` method in a WAX program, you must format the statement correctly, otherwise it could cause your Web server crash. In particular, make sure you do not pass in more format specifiers (for example, "%s" conversion flags) than there are parameters to fill them.

Request Data

Data associated with a URI request is stored in a hash table called `ct_request_data`. This table is passed between phase handlers. The `ct_request_data` structure is located in the `ct_request_data.h` header file.

This hash table contains certain values that are constants and certain values that are dynamically allocated. If a WAX chooses to modify a dynamically allocated value, it must manage the memory associated with that value. When replacing a dynamic value, you must free the old memory. When adding a dynamic value, you must allocate it using `ct_malloc` or `ct_strdup`. In Table 8.4, the "Dynamic?" column indicates which values are dynamic.

The request data is passed to the phase handlers and is also directly available to functions other than phase handlers through the code:

```
ct_get_request_data_table (void* request)
```

The input parameter `request` is a pointer to the Web server dependent structure. The value of each request is described in the following table:

Table 8.3 Values of Input Parameter Request

Request	Pointer to
iPlanet Web server	iPlanet Request structure, <code>request*</code>
IIS	Filter Context structure, <code>PHTTP_FILTER_CONTEXT</code>



Note: For IIS, the `ct_request_data` is not available until the `SF_NOTIFY_URL_MAP` phase.

Request data is retrieved from the request data table through the hash lookup using the keys described here in Table 8.4:

Table 8.4 RSA ClearTrust Web Server Agent Request Data

Key	Type	Dyna-mic?	Value
CT_ALLOWABLE_AUTH_MODES	char *	Yes	The allowable authentication types for the current request. The Authentication handler determines what types of authentication types are accepted for the current URI request: BASIC, CERTIFICATE, CUSTOM, NT, SECURID.
CT_AUTH_MODE	char *	No	The Authorization mode. The Authorization handler determines what type of authorization to perform using this value. The values are: UPW - Perform Authentication and Authorization. User ID and password are supplied. UDN - Perform Authentication and Authorization. Locate user by Distinguished Name. UID, UNT, CUSTOM, SECURID—Perform Authorization only. Locate user by user ID.
CT_AUTHENTICATED	unsigned int	No	A bit mask that specifies the types of authentication the user is currently authenticated against: CT_AUTH_BASIC (0x00000001) CT_AUTH_CERTIFICATE (0x00000004) CT_AUTH_CUSTOM (0x00010000) CT_AUTH_NT (0x00000002) CT_AUTH_SECURID (0x00000008)
CT_DN	char *	Yes	The user's Distinguished Name. When the CT_AUTH_MODE is set to UDN, this value is used to retrieve the Distinguished Name for authorization.
CT_ERR_MSG	char *	No	The error message to return to the browser. When a WWW-AUTHENTICATE (HTTP/401), FORBIDDEN (HTTP/403), or SERVER ERROR (HTTP/500) is returned to the browser, this message is included.

Table 8.4 RSA ClearTrust Web Server Agent Request Data

Key	Type	Dyna- mic?	Value
CT_FORM_AUTH_MODE	char *	Yes	The authentication mode of the form. This value specifies which authentication mechanism to use to authenticate the data in the form.
CT_IS_PATH_PROTECTED	char *	No	Specifies whether or not the URL is protected (Yes/No).
CT_PASSWORD	char *	Yes	The user's password. When the CT_AUTH_MODE is set to UPW, this value is used to retrieve the user's password for authentication.
CT_QUERY	char *	Yes	Query-string portion of the requested URI.
CT_POST_DATA	char *	No	Raw form data associated with a form-based logon request.
CT_ORIG_URI	char *	No	Original URI requested before the user was redirected to a logon page.
CT_PREV_USER	char *	Yes	The user ID or DN for the previous authentication.
CT_STATUS	int	No	The status. This value is used by the status handler to determine the Web server action and/or which handler to execute.
CT_URI	char *	Yes	The requested URI. IF this value is overridden, the Web server will be instructed to serve the new URI.
CT_PLUGIN_USER	char *	Yes	The user's ID. When the CT_AUTH_MODE is set to UPW, this value is used to retrieve the user ID for authentication. For both UPW and UID, this value is used for authorization.
CT_USER_DATA	void *	No	A pointer to a buffer containing user-defined raw data included with the RSA ClearTrust cookie. The length of the buffer is specified in the CT_USER_DATA_LEN status code.
CT_USER_DATA_LEN	unsigned short	No	The length of the CT_USER_DATA buffer, in bytes. The maximum length is 2048 bytes.



Note: You can configure the Web server to return a customized HTML page with the HTTP return codes

Status Handler

The RSA ClearTrust Agent has a single status handler that manages the processing flow based on the status code returned from any phase handler. The status handler does *not* know the identity of the phase handler that invokes it; it simply uses the status code to determine the next action to take or the next phase handler to execute.

The status code is set in `ct_request_data`. For details about `ct_request_data`, refer to the section titled “Request Data” Table 8.5 lists the recognized status codes and their resulting actions.

The status code determines the action and/or the next phase of execution. For convenience, two macros — `SET_STATUS` and `GET_STATUS` — are supplied to set and get the status respectively. Table shows the recognized values and meanings of the different status codes. The status code values are found in the `ct_function_table.h` header file.

Table 8.5 Recognized Status Codes and Resulting Actions

Status Code	Resulting Actions
CT_SESSION_ACTIVE	Execute authentication phase with active authentication.
CT_AUTH_URL_PROTECTED	Execute session phase.
CT_CHECK_ACCESS_REQUIRED	Execute Authorization phase.
CT_CREATE_COOKIE	Execute the Cookie phase.
CT_AUTH_URL_ACCESS_ALLOWED CT_AUTH_URL_UNPROTECTED	Return the requested URI to the browser.
CT_AUTH_BAD_USERNAME CT_AUTH_BAD_PASSWORD	Return a WWW-Authenticate (HTTP 401) to the browser. For form-based authentication, the session is invalidated, and the user is redirected to a logon page.
CT_SESSION_EXPIRED	Force new user authentication.
CT_AUTH_PASSWORD_EXPIRED_FORCED CT_AUTH_PASSWORD_EXPIRED_NEW_USER	Force password change.
CT_AUTH_EXPIRED_ACCOUNT CT_AUTH_INACTIVE_ACCOUNT CT_AUTH_PASSWORD_EXPIRED CT_AUTH_URL_ACCESS_DENIED CT_AUTH_USER_LOCKED_OUT	Return a FORBIDDEN (HTTP 403) to the browser. For form-based authentication, the user is redirected to the appropriate error page.
CT_AUTH_UNKNOWN_ERROR CT_AUTH_DATABASE_ERROR CT_COOKIE_ERROR CT_NO_AUTH_SERVERS CT_SERVER_TIMED_OUT CT_UNHANDLED_REQUEST	Return an error message (HTTP/500) to the browser. For form-based authentication, the user is redirected to the appropriate error page.

Loading Parameter Settings

When the Agent calls `ct_wax_init()` to initialize your WAX program, it passes a pointer (the `config` parameter to `ct_wax_init()` as shown on page 227) to a table containing the Agent's configuration parameters, expressed as name/value pairs. These are the parameter settings loaded from the Agent's `webagent.conf` file.

Your WAX program can use the `ct_table_find()` method (see `ct_table.h`) to get the value of any parameter by name. You can add custom parameters to the `webagent.conf` and access them here. Each custom parameter you add may have any name, provided it does *not* begin with "cleartrust.agent", which is the reserved RSA ClearTrust prefix.

If your Web server uses virtual hosts, you should note the scoping rules in "Scope of WAX Parameters" in the section that follows.

Using WAX Programs with Virtual Host-Enabled Servers

General Issues

To apply a WAX program to a single virtual host, you must declare that WAX in the `<VirtualHost ...>` block for that virtual host. This setting is done in the Agent's `webagent.conf` file. You may apply the same WAX program to many hosts by declaring it in the `<VirtualHost ...>` blocks for all hosts that will use that WAX program.



Warning: If you declare a WAX in any `<VirtualHost ...>` block, you may *not* declare that WAX in the `<Global>` block of the `webagent.conf`.

Scope of WAX Parameters

When a WAX is running within the context of a virtual host, it has access to the `webagent.conf` parameters for that virtual host. These parameters include all the settings from the applicable `<VirtualHost ...>` block plus, for each value not defined in that block, the setting from the `<Global>` block. See "Loading Parameter Settings" above for information on retrieving parameter values.

9

Customizing Your Web Environment

This chapter provides information about various customizations and personalizations that you can implement in your RSA ClearTrust®-protected Web servers. Many of these are provided as samples to get you started developing and creating your own customized and personalized forms and Web server applications.



Note: In this chapter, the RSA ClearTrust Web Server Agent installation directory is referred to as <CT_AGENT_ROOT>. Directories that contain other software are abbreviated in a similar way, such as <IPLANET_SERVER_DIR> and <APACHE_SERVER_DIR>.

Personalizing the Environment

When users attempt to access an RSA ClearTrust-protected resource, they'll see either the default browser-based Basic authentication prompt or one of the HTML forms provided with the RSA ClearTrust installation. You can easily customize the existing HTML pages to conform to your enterprise-wide Web design standards or create new pages, as long as you include the necessary inputs for logon, such as *user* and *password*.

```
<INPUT TYPE="text" NAME="user">  
<INPUT TYPE="password" NAME="password">
```

Modifying the built-in forms is the simplest way to get started with customizing the environment for your users. A more sophisticated approach is to use the RSA ClearTrust APIs to create your own application that generates different home pages based on the user account of the person who logs on.



Warning: When implementing **error pages**, you should be careful to limit the amount of information made available to a possible attacker.

In the event of a failure, whether it be during or after authentication, your error messages or pages should reveal as little information as possible. Logon failures should only be reported as failures; if you display an error page indicating an invalid password was typed, that will tell attacker that he has found a valid user name.

Also, you should avoid displaying the "ClearTrust" name in error messages; the knowledge that a site is secured with RSA ClearTrust can be vital to an attacker should any vulnerabilities be discovered in the RSA ClearTrust product. Every step should be taken to make RSA ClearTrust transparent to the end-user and potential intruders.

Creating Personalized Content

Every virtual enterprise network has different needs. With the RSA ClearTrust system, you can easily customize the RSA ClearTrust Java modules to create dynamically-generated Web pages that contain content or provide a menu of applications that a user is entitled to access, based on the user's identity.

Once you've enabled the RSA ClearTrust Web Server Agent on a Web server, the user name is passed as the `REMOTE_USER`, `CT_REMOTE_USER` or `HTTP_REMOTE_USER` variable (after the user has authenticated), in the HTTP header. You can write a CGI application or Servlet that retrieves this environment variable and displays a user-specific menu of applications or a Web page that has been tailored to that particular user (for example, with user preferences you have stored in the RSA ClearTrust database). You use the RSA ClearTrust API to create this personalization application. See the earlier chapters in this book more information about developing applications using the RSA ClearTrust APIs.



Note: `UNMAPPED_REMOTE_USER` is an IIS-specific environment variable

RSA ClearTrust Environment Variables

The RSA ClearTrust Web Server Agent maintains custom RSA ClearTrust environment variables that identify the type of authentication that was used and the UID for the current user. That means that you (webmaster, application developer) can retrieve this information, just as you do any other existing environment variables on your Web server.

You can access three different environment variables in CGI scripts to identify the authenticated user:

- `REMOTE_USER` (on a proxy server),
- `HTTP_REMOTE_USER` (on a proxy server), or
- `CT_REMOTE_USER`.

Also, you can retrieve `CT_WEB_SVR_ID`, which is the name of the Web server as defined in the RSA ClearTrust Entitlements data store.

If you have written RSA ClearTrust Agent extensions using the standard `REMOTE_USER` environment variable, in 4.5.x and later there is an RSA ClearTrust `CT_REMOTE_USER` environment variable. RSA ClearTrust sets both the `REMOTE_USER` and the `CT_REMOTE_USER` variables, however RSA Security recommends using the `CT_REMOTE_USER` variable to avoid other applications from overwriting the commonly used `REMOTE_USER`.

Details

The Web Server Agent maintains an environment variable named `CT_REMOTE_USER`. After a successful authentication, this environment variable sets the UID for the current user. The Agent will also set the standard `REMOTE_USER` variable, however sometimes other applications will overwrite `REMOTE_USER`.

RSA Security recommends using `CT_REMOTE_USER`. This variable can be accessed in the same way as the `CT_AUTH_TYPE` environment variable.

The Web Server Agent maintains an environment variable named `CT_AUTH_TYPE`. The variable is defined according to the following bit mask values:

Table 9.1 `CT_AUTH_TYPE` bitmask values

<code>CT_AUTH_TYPE</code>	Bitmask
<code>CT_AUTH_BASIC</code>	0x00000001
<code>CT_AUTH_NT</code>	0x00000002
<code>CT_AUTH_CERTIFICATE</code>	0x00000004
<code>CT_AUTH_SECURID</code>	0x00000008
<code>CT_AUTH_CUSTOM</code>	0x00001000

CGI Usage

To obtain the value of the `CT_AUTH_TYPE` or `CT_REMOTE_USER` in the CGI environment, you must pass in `HTTP_CT_AUTH_TYPE` or `HTTP_CT_REMOTE_USER` as the key. For example, in C, you would:

```
getenv("HTTP_CT_AUTH_TYPE")
getenv("HTTP_CT_REMOTE_USER")
```

While in Perl, you would use:

```
$ENV{HTTP_CT_AUTH_TYPE}
$ENV{HTTP_CT_REMOTE_USER}
```

The key is the same for Apache, Microsoft IIS, and Netscape Enterprise Server, on both UNIX and Windows platforms.

Servlet Usage

Obtaining the `CT_AUTH_TYPE` or `CT_REMOTE_USER` in a Java Servlet depends on the type of Web server that you're using. Specifically, for Apache, you should use `ct_auth_type`. For Microsoft IIS, you must parse the `ALL_HTTP` variable and retrieve `CT_AUTH_TYPE` from `HTTP_CT_AUTH_TYPE` or `CT_REMOTE_USER` from `HTTP_CT_REMOTE_USER`. For Netscape, you retrieve `ct-auth-type` or `ct_remote_user`.

Example

Webmasters, developers, and others who are deploying the RSA ClearTrust Web Server Agent can use this environment variable (just as they would any other environment variable) in CGI scripts or servlets. Here is a Servlet example of the logic one might use to determine the type of RSA ClearTrust authentication that was used, based on the bit mask.

```
private static int SECURID_AUTH_MASK = 0x00000008;
private static int CERT_AUTH_MASK   = 0x00000004;
private static int NT_AUTH_MASK     = 0x00000002;
private static int BASIC_AUTH_MASK  = 0x00000001;

private static int SECURITY_LEVEL_1 = 1;
private static int SECURITY_LEVEL_2 = 2;
private static int SECURITY_LEVEL_3 = 3;

String customHeader = req.getHeader("CT_AUTH_TYPE");

// isAuthenticated is the value of CT_AUTH_TYPE set by
// ClearTrust Agent
private int getAuthLevel(int isAuthenticated)
{
    if ( (isAuthenticated & SECURID_AUTH_MASK) > 0 )
    {
        return SECURITY_LEVEL_3;
    }
    else if ( (isAuthenticated & CERT_AUTH_MASK) > 0 )
    {
        return SECURITY_LEVEL_2;
    }
    else if ( (isAuthenticated & BASIC_AUTH_MASK) > 0 ||
              (isAuthenticated & NT_AUTH_MASK) > 0 )
    {
        return SECURITY_LEVEL_1;
    }
    return -1; // invalid auth mode
}
```

RSA ClearTrust Source Code Details

The header file, `ct_request_data.h`, contains the bitmask and variable information. See the `ct_request_data.h` header file, in the `<CT_AGENT_ROOT>/include` directory of the RSA ClearTrust installation, for details. Also see [Chapter 8, “Web Agent Extension API”](#) for additional information about the RSA ClearTrust Web Server Agent Extension (WAX) API and how it works.

Contents of the RSA ClearTrust Cookie

The RSA ClearTrust cookie contains the following pieces of information:

- cookie name, which is CTSESSION by default but may be changed. See the following section entitled [“Changing the Cookie Name”](#).
- user’s RSA ClearTrust user ID name
- length of the user ID
- length of the user’s password
- IP address of the server that issued this cookie, represented as 32 bit address
- IP Address of the client (browser) machine, represented as string
- date and time that this cookie was first issued
- most recent date and time this cookie was updated, also known as the “last touch time”
- boolean indicating whether this cookie is now valid. True indicates it is valid.
- bitmask indicating all the authentication types that the user has satisfied in this session
- for multi-phase RSA SecurID authentication only, a special flag that indicates current RSA SecurID authentication phase
- for multi-phase RSA SecurID authentication only, information identifying which Authorization Server is handling the current authentication
- size of the additional user-defined data buffer
- the user-defined data buffer itself

Changing the Cookie Name

By default, the RSA ClearTrust cookie name is CTSESSION. When you configure the RSA ClearTrust system you may change this name.

Generally, you will only need to change the name if multiple RSA ClearTrust installations are present on a network, or if the cookie name conflicts with another application.

To change the cookie name, edit the `<CT_AGENT_ROOT>/conf/webagent.conf` file and change the `cleartrust.agent.cookie_name` setting. You may use any combination of letters and numerals without spaces. Hyphen and underscore characters are permitted; other non-alphanumeric characters are not allowed. For example, you might set it as follows:

```
cleartrust.agent.cookie_name=RSA_W_EUROPE
```

Note that if SSO is enabled on your RSA ClearTrust installation, you *cannot* leave this setting blank.

All Web servers that are participating in SSO must use the same cookie name.

Writing ASP and JSP Pages

RSA ClearTrust Parameter Names

The following ASP query string parameters may be used in ASP and JSP pages that you write. RSA strongly recommends that you do not change these parameter names:

- `CTAuthMode` is the RSA ClearTrust authentication type, such as BASIC, NT, SECURID, or CUSTOM.

For example, in a JSP you might retrieve the authentication type as follows:

```
String MyAuthMode = request.getParameter("CTAuthMode");
```

- `CTLoginErrorMsg` is the error message string to be displayed to the user.

Password Changer Example

This Administrative API example, `PasswordChanger`, is a Java Server Pages (JSP) example that shows how to build a page that will let a user change his or her password.

A compiled version of this example is provided in your release as a Web application archive (WAR) file deployable on an application server:

```
<CT_HOME>/api/admin-j/example/changepw.war
```

Using the Password Changer

You can configure RSA ClearTrust to launch the password changer automatically when a user enters an expired password. See the section explaining the `cleartrust.agent.login_error_password_expired` parameter in Appendix A of the RSA ClearTrust Installation and Configuration Guide.

Deploying the Example

This section describes how to deploy the Password Changer example WAR file, `changepw.war`, on a JRun application server. Installation procedures for other application servers will differ somewhat from the steps below.

1. Open the JRun Management Console and click the name of your application server. By default, this is **JRun Default Server**.
2. Click **WAR Deployment** and enter appropriate values in the following fields.
 - **Servlet WAR File or Directory:** Enter the full path to the `changepw.war` file, which is usually `<CT_HOME>/api/admin-j/example/changepw.war`

- **JRun Server Name:** Select the name of your application server. By default, this is **JRun Default Server**.
- **Application Name:** Enter a unique name for the application, without spaces between characters, for example, **changepw**.
- **Application Host:** Select **All Hosts**.
- **Application URL:** Enter the URL to which the Entitlements Manager will be mapped. This must begin with a forward slash, and should be an easily remembered name like `/changepw`
- **Application Deployment Directory:** Enter the root directory where the application will be stored and served in JRun. A typical installation deployment directory is as follows, where `<JRUN_HOME>` is the path location of your JRun installation directory:
`<JRUN_HOME>/servers/default/changepw`

3. Click **Deploy**.

4. Restart your JRun Application Server to complete the deployment.

Now that the WAR file is deployed, you can run it by pointing your browser to the Application URL you specified above. For example, if you installed on your local machine and your JRun Server port is 8100, then the URL might be:

```
http://localhost:8100/changepw
```

Customizing the Example

The source code is also provided in your installation, in the following files:

- `<CT_HOME>/api/admin-j/example/PasswordChanger.java`
- `<CT_HOME>/api/admin-j/example/index.jsp`
- `<CT_HOME>/api/admin-j/example/success.jsp`

To modify this sample application, follow these steps:

- Edit the source files.
- If you have changed `PasswordChanger.java`, then use `compile.bat` or `compile.sh` to compile the new program. This will compile a new `PasswordChanger.class`, but it will not update the `changepw.war` archive.
- Use an archive tool such as WinZip or “`jar -u`” to edit the `changepw.war` archive, replacing the files that you edited (one or more of the following: `PasswordChanger.class`, `index.jsp`, `success.jsp`) and adding any new files.

Example Source

PasswordChanger.java:

```
package sirrus.samples.admin;

import java.io.IOException;

import sirrus.api.client.APIServerProxy;
import sirrus.api.client.APIException;
import sirrus.api.client.TransportException;
import sirrus.api.client.ObjectNotFoundException;
import sirrus.api.client.IllegalPasswordException;

/**
 * ChangePassword bean that encapsulates the business logic that
 * handles setting up an RSA ClearTrust API Server connection
 * and doing the business logic mechanics of changing a user's
 * password in the ClearTrust backend. This is the main meat of
 * the web-based default change password mechanism.
 *
 * @since ClearTrust 4.5 SE P1
 * @version 1.0
 */
public class PasswordChanger
{
    /**
     * The display string to show to the user after the
     * operation completes. This is fetched with a separate
     * method call after the changePassword method is
     * called.
     */
    private String displayString;

    /**
     * The user name that we are currently operating on.
     */
    private String userName;

    /**
     * The old user password that the user is changing their
     * password FROM.
     */
    private String oldPassword;
```

PasswordChanger example continues:

```
/**
 * The new password the user is changing their password
 * TO.
 */
private String newPassword;

/**
 * The confirmation of the new password. This is used to
 * verify the user's password before it is set.
 */
private String confirmNewPassword;

/**
 * The hostname of the RSA ClearTrust API Server to
 * connect to.
 */
private String apiServerHost;

/**
 * The port that the RSA ClearTrust API Server is running
 * on.
 */
private int apiServerPort;

/**
 * Is the RSA ClearTrust API Server using ssl?
 */
private boolean sslEnabled;

/**
 * The business method to change the password. This does
 * input validation, and then if everything is ok, it
 * attempts to change the password on the server back-end. If
 * the method returns false, the reason can be retrieved
 * through the getDisplayString method.
 *
 * @return true if the password is changed, false otherwise
 * @see #getDisplayString
 */
public boolean changePassword()
{
```

PasswordChanger example continues:

```

if ((apiServerHost == null) || (apiServerPort == 0))
{
    displayString = "Configuration Error: API server " +
        "unspecified.";
    return false;
}

if ((userName == null) || (oldPassword == null) ||
    (newPassword == null) ||
    (confirmNewPassword == null))
{
    displayString = "Please enter username and " +
        "password";
    return false;
}

if (!(newPassword.equals(confirmNewPassword)))
{
    displayString = "Passwords do not match";
    return false;
}

// Connect to the backend.
APIServerProxy serverProxy = null;
try
{
    serverProxy = new APIServerProxy(apiServerHost,
                                    apiServerPort,
                                    sslEnabled);

    // This is the way you connect in anon mode
    // for a password change.
    serverProxy.connect(null,null,null,null);

    serverProxy.resetPassword(userName,
                              oldPassword,
                              newPassword);
    displayString = "Password successfully changed";
    return true;
}

```


PasswordChanger example continues:

```
catch (TransportException te)
{
    displayString = "Error connecting to server: " +
        te.getMessage();
    return false;
}

catch (ObjectNotFoundException onf)
{
    displayString = "Login incorrect";
    return false;
}

catch (IllegalPasswordException ipe)
{
    displayString = "New password not allowed by " +
        "password policy";
    return false;
}

catch (IOException io)
{
    displayString = "Error connecting to server: " +
        io.getMessage();
    return false;
}

catch (APIException api)
{
    displayString = "Error in server: " +
        api.getMessage();
    return false;
}
}
```

PasswordChanger example continues:

```

/**
 * Get the display string to show to the user after the
 * operation is completed.
 *
 * @return the user display string
 */
public String getDisplayString()
{
    return displayString;
}

/**
 * The public bean method for setting the RSA ClearTrust
 * API Server host.
 *
 * @param host The hostname of the RSA ClearTrust API
 *             Server to use
 */
public void setServerHost(String host)
{
    apiServerHost = host;
}

/**
 * Set the RSA ClearTrust API Server port. This is the port
 * used to connect to the RSA ClearTrust API Server to
 * change the user's password.
 *
 * @param port The TCP port of the RSA ClearTrust API
 *             Server
 */
public void setServerPort(int port)
{
    apiServerPort = port;
}

```

PasswordChanger example continues:

```
/**
 * Is the server using ssl? This must match the other global
 * settings of the system.
 *
 * @param ssl True if the system is in ssl mode, false
 *           otherwise
 */
public void setSslEnabled(boolean ssl)
{
    sslEnabled = ssl;
}

/**
 * Set the name of the user whose password is to be changed.
 *
 * @param user The name of the user
 */
public void setUsername(String user)
{
    userName = user;
}

/**
 * Get the username of the user whose password is being
 * changed.
 *
 * @return the username whose password is being changed,
 *         or "" if unset
 */
public String getUsername()
{
    if (userName != null)
        return userName;
    else
        return "";
}
```

PasswordChanger example continues:

```
/**
 * Set the old password of the user. This is used as the
 * credential to allow the user to change their password.
 *
 * @param pw The old (current) user password
 */
public void setOldPassword(String pw)
{
    oldPassword = pw;
}

/**
 * Set the new password of the user. This will be the
 * new password of the user if the operation completes.
 *
 * @param pw The new value of the password
 */
public void setNewPassword(String pw)
{
    newPassword = pw;
}

/**
 * Set the confirmation of the new password. This is used to
 * verify that the user entered the correct username and
 * password.
 *
 * @param pw The confirmation of the new password
 */
public void setConfirmPassword(String pw)
{
    confirmNewPassword = pw;
}
}
```

In addition to the PasswordChanger class, two JSPs are required for this example. The first is index.jsp, shown here:

```
<%@page contentType="text/html"%>

<%--

This is an example jsp that allows the user to change their own password.
This page is intended to be used as a starting point for creating a page
that is customized and suited for a particular ClearTrust deployment.

--%>

<jsp:useBean id="passwordChanger" class="sirrus.samples.admin.PasswordChanger"
scope="session"/>

<%--

Set all of the input parameters in the password changer. This is done before
the serverHost, serverPort and sslEnabled values are explicitly set so that
these parameters -cannot- be set via the query string.

--%>

<jsp:setProperty name="passwordChanger" property="*/>

<%--

These three parameters must be set in the servlet context init to the
hostname and port of the user's administrative api server. For example:

    <context-param>
        <param-name>passwordChanger.serverHost</param-name>
        <param-value>localhost</param-value>
    </context-param>
    <context-param>
        <param-name>passwordChanger.serverPort</param-name>
        <param-value>5601</param-value>
    </context-param>
    <context-param>
        <param-name>passwordChanger.sslEnabled</param-name>
        <param-value>>false</param-value>
    </context-param>

The default values (show above) will be used if the parameters are not
provided.

--%>
```

index.jsp (part of PasswordChanger example) continues:

```
<%
// get the init parameters from the implicit 'application' variable
// (which is pageContext.getServletContext())
// NOTE: the 'sslEnabled' parameter is a string
String serverHost = application.getInitParameter("passwordChanger.serverHost");
String serverPort = application.getInitParameter("passwordChanger.serverPort");
String sslEnabled = application.getInitParameter("passwordChanger.sslEnabled");

// getInitParameter() will return null if the parameter was not specified.
// if null then use default values
if(serverHost == null)
    serverHost = "localhost";
if(serverPort == null)
    serverPort = "5601";
if(sslEnabled == null)
    sslEnabled = "false";

// set the corresponding properties in the passwordChanger bean
passwordChanger.setServerHost(serverHost);
passwordChanger.setServerPort(Integer.valueOf(serverPort).intValue()); //
convert from string to int
passwordChanger.setSslEnabled(Boolean.valueOf(sslEnabled).booleanValue()); //
convert from string to boolean

%>

<%
// if the password change was successful then display the success page
// if not, then display the form. The first time through this form will
// return false since the username, password, etc are null and the form
// will be displayed.
if (passwordChanger.changePassword())
{
%>

<jsp:forward page="success.jsp"/>

<%
}
else
{
%>
```

index.jsp (part of PasswordChanger example) continues:

```
<html>
  <head>
    <title>
      Change ClearTrust Password
    </title>
  </head>
  <body>
    <h1><jsp:getProperty name="passwordChanger"
property="displayString"/></h1>
    <form method="POST" action="<%=request.getRequestURI()%>">
      <table>
        <tr>
          <td>Name</td>
          <td><input type="text" name="userName"
value="<jsp:getProperty name="passwordChanger" property="userName"/>"></td>
        </tr>
        <tr>
          <td>Current Password</td>
          <td><input type="password" name="oldPassword"></td>
        </tr>
        <tr>
          <td>New Password</td>
          <td><input type="password" name="newPassword"></td>
        </tr>
        <tr>
          <td>Confirm New Password</td>
          <td><input type="password" name="confirmPassword"></td>
        </tr>
        <tr>
          <td colspan="2"><input type="submit" value="Change
Password"></td>
        </tr>
      </table>
    </form>
  </body>
</html>
<% } %>
```

Finally, the PasswordChanger example requires one other jsp file: `success.jsp`, shown here:

```
<%@page contentType="text/html"%>
<%--

This is the page shown when the user successfully completes a password
change

--%>

<jsp:useBean id="passwordChanger" class="sirrus.samples.admin.PasswordChanger"
scope="session"/>

<html>
  <head>
    <title>Password Successfully Changed</title>
  </head>
  <body>
    <h1><jsp:getProperty name="passwordChanger"
property="displayString"/></h1>
  </body>
</html>
```

HTTP Header Parameters

The parameters listed here can be set in the Web Agent's `webagent.conf` file to specify what information is available as HTTP request headers.

For the 4.7 release, `CTUSER` replaces the old `CT_REMOTE_USER` variable:

```
cleartrust.agent.user_header_list=CTUSER
```

The following parameters can be set to “yes” or “no” to export ClearTrust values to the HTTP header:

```
cleartrust.agent.export_session_init_time=yes  
cleartrust.agent.export_session_expiration_time=yes  
cleartrust.agent.export_last_touch_time=yes  
cleartrust.agent.export_cookie_user_buffer=yes
```

See the *RSA ClearTrust Installation and Configuration Guide* or the comments in the `webagent.conf` file for an explanation of each parameter.

Index

A

- add_admin: 25
- add_admin_user: 25
- add_app: 25
- add_group: 25
- add_realm: 25
- add_server: 25
- add_user: 25
- add_user_prop_def: 25
- addAdministrativeUser: 82
- addUser: 82
- Admin API
 - Java
 - connection types: 72
- admin_id: 25
- Administrative API
 - C: 11
 - connection types: 17
 - Java: 67
- administrative group
 - in Java API: 20
 - object in Java API: 78
 - search for in Java API: 107
- administrative role
 - object in C API: 25
 - object in Java API: 82
- administrative user
 - example code, C: 60
 - object in C API: 24
 - object in Java API: 80
- administrator permissions: 25
- Agent
 - augment vs. override handler: 213
 - edit cookie: 218
 - example code: 222
 - extending functionality: 204
 - extensions, compiling: 216
 - handler flow diagram: 211
 - HTTP header parameters: 253
 - libraries: 213
 - process flow diagram: 206
- agent.auth_resource_list: 158
- agent.cookie_name: 239
- agent.login_error_password_expired: 240
- allocation
 - utilities: 54, 56
- APACHE_SERVER_DIR: 204
- API
 - installing on Solaris: 9
 - installing on Windows: 7
 - WAX: 203
- API Server
 - vs. Entitlements Server: 1
- APIFactory: 171
- APIFactory methods: 171
- APIServerProxy: 69
 - connect method example: 73
 - defined: 69
 - disconnect method example: 74
 - Java code example: 73
- application
 - object in C API: 42
 - object in Java API: 96
 - search for in Java API: 107
- application function
 - object in C API: 44
 - object in Java API: 97
- application server: 240
- application URL
 - object in C API: 44, 99
- array
 - of ClearTrust C objects: 18
 - of ClearTrust Java objects: 102
- ASP
 - example: 197
 - get user example: 201
 - Runtime API example: 198
 - writing ASP pages: 240
- auth_resource_list: 158, 207
- authenticate: 168
 - certificate authentication: 158
 - example code: 172
 - example code with SSL on: 177
- authenticated connection
 - Admin C API: 17
 - Admin Java API: 72
 - Runtime C API: 136
 - Runtime Java API: 162
- authenticated SSL
 - running C Runtime API with: 136
- authentication: 221
 - auth type for a given resource: 158
 - custom: 215, 221
 - custom auth in C Admin API vs. WAX: 148
 - error pages example: 225
 - NT auth example code: 218, 221

- SecurId example code, C: 153
- SecurId example code, Java: 183
- tokens: 140, 159
- tokens in C Runtime API: 133
- type: 158
- types: 132, 158, 169
- authentication handler: 208
- authentication type
 - setting for URI: 207
- authorization handler: 209
- Authorization Server
 - checking with C API: 151
 - clear all caches: 70
 - clear cache: 168
- authorize: 168
- AuthTypes interface: 169
- AuthTypesClass for DCOM: 196

B

- basic authentication
 - C Runtime API: 145
- basic entitlement
 - C API: 36
 - in Java API: 94
- book conventions: ix
- boolean criterion: 105
- Bsymbolic: 217

C

- C API
 - JNI wrapper: 13
 - memory management: 54, 56
 - Runtime: 131
- cache
 - cache management example code: 112
 - clearing all: 70
 - clearing one: 168
- callback
 - for keystore passphrase: 140
 - for private key passphrase: 140
- CD
 - finding APIs on CD: 7
- certificate
 - CT_ROOT: 165
 - keystore name: 140
- certificate authentication
 - C Runtime API: 146
 - not supported in Runtime API: 158
- certj.jar: 13
 - for DCOM API: 190

- for Java API: 68, 161
- change password
 - example code: 240
- changepw.war: 240
- checkAccess: 69
- checkAddGroupToRealm: 103
- checkAddUserToGroup: 103
- checkChangePassword: 103
- checkCreateAdministrativeGroup: 103
- checkCreateAdministrativeRole: 103
- checkCreateApplication: 103
- checkCreateApplicationFunction: 103
- checkCreateExplicitEntitlement: 103
- checkCreateGroup: 103
- checkCreatePasswordPolicy: 103
- checkCreateRealm: 103
- checkCreateServerTree: 103
- checkCreateSmartRule: 103
- checkCreateUser: 103
- checkCreateUserPropertyDefinition: 103
- checkCreateWebServer: 103
- checkDeleteAdministrativeGroup: 103
- checkDeleteAdministrativeRole: 103
- checkDeleteApplication: 103
- checkDeleteApplicationFunction: 103
- checkDeleteExplicitEntitlement: 103
- checkDeleteGroup: 103
- checkDeletePasswordPolicy: 103
- checkDeleteRealm: 103
- checkDeleteServerTree: 103
- checkDeleteSmartRule: 103
- checkDeleteUser: 103
- checkDeleteUserPropertyDefinition: 104
- checkDeleteWebServer: 104
- checkFunction: 69
- checkModifyAdministrativeGroup: 104
- checkModifyAdministrativeRole: 104
- checkModifyApplication: 104
- checkModifyApplicationFunction: 104
- checkModifyExplicitEntitlement: 104
- checkModifyGroup: 104
- checkModifyPasswordPolicy: 104
- checkModifyRealm: 104
- checkModifyServerTree: 104
- checkModifySmartRule: 104
- checkModifyUser: 104
- checkModifyUserPropertyDefinition: 104
- checkModifyWebServer: 104
- checkPassword: 69
- checkRemoveGroupFromRealm: 104
- checkRemoveUserFromGroup: 104

checkResourceStatus: 168
 checkViewPasswordPolicy: 104
 CLASSPATH
 for DCOM API: 191
 clearServerCaches: 168
 cleartrust.agent.auth_resource_list: 158, 207
 cleartrust.agent.cookie_name: 239
 cleartrust.agent.export_cookie_user_buffer: 253
 cleartrust.agent.export_last_touch_time: 253
 cleartrust.agent.export_session_expiration_time: 253
 cleartrust.agent.export_session_init_time: 253
 cleartrust.agent.login_error_password_expired: 240
 cleartrust.agent.pix directive: 227
 cleartrust.agent.user_header_list: 253
 cleartrust.agent.wax: 212, 213
 cleartrust.agent.wax directive: 227
 cleartrust.aserver.ldapauth.ldapattr_map_scuid: 147
 cleartrust.dispatcher.list_port.force_anonymous: 136
 clearUserPropertyCriterion: 109
 client key: 141
 close: 168
 code
 error in C API: 54
 collection
 C API and: 18
 sparse data: 102
 com
 package: 196
 COM applications
 using with RSA ClearTrust: 189
 compile
 Agent: 216
 compiling and linking
 for Microsoft IIS: 216
 for Netscape/NT: 216
 for Netscape/UNIX: 216, 217
 configuration
 custom parameters for WAX: 233
 connect: 69
 Admin C API connection types: 17
 Admin Java API code example: 73
 Admin Java API connection types: 72
 Runtime C API connection types: 136
 Runtime Java API connection types: 162
 connection pool
 C Runtime functions: 138
 defined: 135
 keys in C API: 140
 constants
 C API: 54
 contact information: xi
 conventions: ix
 cookie
 contents: 239
 editing: 218
 name: 239
 name, changing: 239
 WAX example: 218
 cookie handler: 210
 createAdministrativeGroup: 69
 createAdministrator: 78
 createApplication: 69
 createApplicationFunction: 96
 createApplicationURL: 96
 createAppURL: 69
 createAuthServerConnection: 159, 160, 171
 createAuthServerConnection(): 171
 createExplicitEntitlement: 69
 createFromServerDispatcher: 171
 createFromServerDispatchers: 171
 createFromServerList: 171
 createGroup: 69
 createPasswordPolicy: 69
 createRealm: 69
 createServerTree: 100
 createSmartRule
 IApplication method: 96
 IApplicationFunction method: 98
 IApplicationURL method: 99
 createToken: 168
 createUser: 69
 createUserPropertyDefinition: 69
 createWebApplication: 70
 createWebAppURL: 70
 createWebServer: 70
 CredentialConstants: 170
 criteria
 Java API classes: 105
 ct_add_group_to_realm: 35
 ct_add_server_to_pool_ext: 139
 ct_add_server_to_pool_ext_v2: 139
 ct_add_user_to_admin_role: 26
 ct_add_user_to_group: 29
 CT_Admin object: 25
 ct_admin_api.jar
 DCOM and: 190
 ct_admin_api.lib and dll: 13
 CT_AdminUser: 24
 vs. CT_User: 24
 CT_AGENT_ROOT: 204
 ct_alloc_array_obj: 54
 ct_alloc_criterion: 54

ct_alloc_obj: 54
 ct_alloc_search: 49, 54
 ct_alloc_string: 54
 ct_alloc_userprop_criterion: 54
 ct_alloc_userprop_criterion_array: 54
 CT_ALLOWED_AUTH_MODES: 230
 CT_Application: 42
 CT_ApplicationFunction: 44
 CT_ApplicationURL: 44
 CT_AUTH_ADMIN_LOCKOUT_STR: 145, 146
 CT_AUTH_BAD_PASSWORD: 208, 232
 CT_AUTH_BAD_USERNAME: 208, 232
 CT_AUTH_CUSTOM: 208, 214
 CT_AUTH_DATABASE_ERROR: 232
 CT_AUTH_EXPIRED_ACCOUNT: 209, 232
 CT_AUTH_EXPIRED_ACCOUNT_STR: 145, 146
 CT_AUTH_EXPIRED_PASSWORD_FORCED_STR: 145
 CT_AUTH_EXPIRED_PASSWORD_NEW_USER_STR: 145
 CT_AUTH_EXPIRED_PASSWORD_STR: 145, 146
 CT_AUTH_INACTIVE_ACCOUNT: 209, 232
 CT_AUTH_INACTIVE_ACCOUNT_STR: 145, 146
 CT_AUTH_INVALID_PASSWORD_STR: 145
 CT_AUTH_MODE: 208, 230
 CT_AUTH_NEW_PIN_ACCEPTED_STR: 147
 CT_AUTH_NEW_PIN_REJECTED_STR: 147
 CT_AUTH_NEW_PIN_REQUIRED_STR: 146, 147
 CT_AUTH_NEXT_CODE_REQUIRED_STR: 147
 CT_AUTH_NT_AUTH_FAILED_STR: 146
 CT_AUTH_PASSWORD_EXPIRED: 209, 232
 CT_AUTH_PASSWORD_EXPIRED_FORCED: 209, 232
 CT_AUTH_PASSWORD_EXPIRED_NEW_USER: 209, 232
 ct_auth_result structure: 134
 ct_auth_result.h: 13, 134
 ct_auth_result_tostring: 134
 CT_AUTH_SECURID_AUTH_FAILED_STR: 147
 CT_AUTH_SERVER_STATE: 147
 CT_AUTH_TYPE: 237
 ct_auth_types.h: 134
 CT_AUTH_UNKNOWN_ERROR: 232
 CT_AUTH_UNKNOWN_USER_STR: 145
 CT_AUTH_URL_ACCESS_ALLOWED: 209, 232
 CT_AUTH_URL_ACCESS_DENIED: 209, 232
 CT_AUTH_URL_PROTECTED: 232
 CT_AUTH_URL_UNPROTECTED: 232
 CT_AUTH_USER_LOCKED_OUT: 209, 232
 CT_AUTH_VALID_USER_STR: 145, 146
 ct_authenticate: 150
 ct_authenticate_pool: 150
 CT_AUTHENTICATED: 208, 230
 ct_authorize: 149
 ct_authorize_pool: 149
 ct_bool.h: 12
 ct_boolean.h: 13, 134
 ct_callback.h: 134
 CT_CALLBACK_KEYSTORE_PASSPHRASE: 140
 CT_CALLBACK_PRIVATE_KEY_PASSPHRASE: 140
 ct_change.html, replacement for: 240
 ct_check_access (deprecated): 19
 CT_CHECK_ACCESS_REQUIRED: 232
 ct_check_add_group_to_realm: 52
 ct_check_add_user_to_group: 52
 ct_check_change_password: 52
 ct_check_create_admin_group: 52
 ct_check_create_admin_role: 52
 ct_check_create_admin_role_in_admin_group: 52
 ct_check_create_application: 52
 ct_check_create_explicit_entitlement: 52
 ct_check_create_group: 52
 ct_check_create_password: 52
 ct_check_create_property_definition: 52
 ct_check_create_realm: 52
 ct_check_create_server_tree: 52
 ct_check_create_smart_rule: 52
 ct_check_create_user: 52
 ct_check_create_web_server: 52
 ct_check_create_application_function: 52
 ct_check_delete_administrative_group: 52
 ct_check_delete_administrative_role: 52
 ct_check_delete_application: 52
 ct_check_delete_application_function: 52
 ct_check_delete_explicit_entitlement: 52
 ct_check_delete_group: 52
 ct_check_delete_password_policy: 52
 ct_check_delete_realm: 52
 ct_check_delete_server_tree: 52
 ct_check_delete_smart_rule: 53
 ct_check_delete_user: 53
 ct_check_delete_user_property_definition: 53
 ct_check_delete_web_server: 53
 ct_check_function (deprecated): 19
 ct_check_modify_administrative_group: 53
 ct_check_modify_administrative_role: 53
 ct_check_modify_application: 53
 ct_check_modify_application_function: 53
 ct_check_modify_explicit_entitlement: 53
 ct_check_modify_group: 53
 ct_check_modify_password_policy: 53

ct_check_modify_realm: 53
ct_check_modify_server_tree: 53
ct_check_modify_smart_rule: 53
ct_check_modify_user: 53
ct_check_modify_user_property_definition: 53
ct_check_modify_web_server: 53
ct_check_password (deprecated): 19
ct_check_resource_status: 150
ct_check_resource_status_pool: 150
ct_check_set_default_password_policy: 53
CT_CLASSPATH_KEY: 15
ct_clear_server_caches: 150
ct_clear_server_caches_pool: 150
ct_clear_three_maps: 152
ct_commands.h: 12
ct_connect: 17, 18
ct_connect_admin: 18
CT_COOKIE_ERROR: 210, 232
ct_create_admin_group: 21
ct_create_admin_role: 26
ct_create_app_func: 44
ct_create_app_function_map: 152
ct_create_app_url: 45
ct_create_application: 43
ct_create_blank_pool: 138, 139
ct_create_blank_pool_ext: 138
CT_CREATE_COOKIE: 232
ct_create_entitlement_for_group: 29, 37
ct_create_entitlement_for_realm: 35, 37
ct_create_entitlement_for_user: 32, 37
ct_create_entitlement_for_user_and_appfunc: 37
ct_create_entitlement_for_user_and_application: 37
ct_create_group: 29
ct_create_map: 151
ct_create_password_policy: 27
ct_create_pool_with_dispatcher: 138
ct_create_pool_with_dispatchers_ext: 133, 138
ct_create_pool_with_dispatchers_ext_v2: 138
ct_create_realm: 35
ct_create_server_tree: 47
ct_create_token: 150
ct_create_token_pool: 150
ct_create_user_and_properties: 32
ct_create_user_basic_map: 152
ct_create_user_dn_basic_map: 152
ct_create_user_dn_ldap_map: 152
ct_create_user_dn_map: 152
ct_create_user_ldap_map: 152
ct_create_user_map: 152
ct_create_user_nt_map: 152
ct_create_user_property_definition: 34
ct_create_user_securid_map: 152
ct_create_user_securid_new_pin_map: 152
ct_create_user_securid_next_code_map: 152
ct_create_web_resource_map: 152
ct_create_web_server: 46
ct_dcom.jar: 190
ct_delete_admin_group: 21
ct_delete_admin_role: 26
ct_delete_app_func: 44
ct_delete_application: 43
ct_delete_appurl: 45
ct_delete_exp_entitlement: 37
ct_delete_group: 29
ct_delete_password_policy: 27
ct_delete_realm: 35
ct_delete_server_tree: 47
ct_delete_user: 32
ct_delete_userpropdef: 34
ct_delete_webserver: 46
ct_destroy_pool: 139
ct_destroy_three_maps: 152
ct_destroy_two_maps: 152
ct_disconnect: 19
CT_DN: 230
CT_EntityHdr: 28
CT_ERR_MSG: 230
ct_error structure: 134
ct_error.h: 13, 134
ct_error_tostring: 134
CT_ExplicitEntitlement: 36
ct_extension_init: 227
ct_flush_cache (deprecated): 19
ct_force_password_expiration: 19
CT_FORM_AUTH_MODE: 231
ct_free: 57, 229
ct_free_adminroleid: 54
ct_free_adminroleid_array: 19
ct_free_array_obj: 54, 56
ct_free_criterion: 54
ct_free_obj: 54, 56
ct_free_seaarch: 49
ct_free_search: 49, 54
ct_free_string: 54
ct_free_userprop_criterion_array: 54
ct_function_table.h: 214
ct_get_admin_group_by_index: 21
ct_get_admin_user_in_admin_role_by_index: 26
ct_get_admingroup_by_index_in_search: 49
ct_get_admingroup_by_name: 21
ct_get_admingroup_by_name_in_search: 49
ct_get_admingroups_by_index: 21

ct_get_admingroups_by_name: 21
 ct_get_admingroups_by_names_in_search: 49
 ct_get_admingroups_by_range: 21
 ct_get_admingroups_by_range_in_search: 49
 ct_get_adminrole_for_admingroup_by_index: 21
 ct_get_adminrole_for_admingroup_by_name: 21, 26
 ct_get_adminroleids_for_user: 19
 ct_get_adminroles_for_admingroup_by_names: 21, 26
 ct_get_adminroles_for_admingroup_by_range: 21, 26
 ct_get_app_by_index: 43
 ct_get_app_by_index_in_search: 49
 ct_get_app_by_name: 43
 ct_get_app_by_name_in_search: 49
 ct_get_app_for_appfunc: 44
 ct_get_appfunc_for_application_by_index: 43
 ct_get_appfunc_for_application_by_name: 43
 ct_get_appfunc_for_entitlement: 37
 ct_get_appfuncs_for_application_by_names: 43
 ct_get_appfuncs_for_application_by_range: 43
 ct_get_application_for_admingroup_by_index: 21
 ct_get_application_for_admingroup_by_name: 21
 ct_get_application_owner: 43
 ct_get_applications_for_admingroup_by_names: 21
 ct_get_applications_for_admingroup_by_range: 21
 ct_get_apps_by_names: 43
 ct_get_apps_by_names_in_search: 50
 ct_get_apps_by_range: 43
 ct_get_apps_by_range_in_search: 50
 ct_get_appurl_for_application_by_index: 43
 ct_get_appurl_for_webserver_by_index: 46
 ct_get_appurls_for_application_by_range: 43
 ct_get_appurls_for_webserver_by_range: 46
 ct_get_context_admin_role: 26
 ct_get_default_password_policy: 27
 ct_get_entitlement: 38
 ct_get_entitlement_for_group: 29
 ct_get_entitlement_for_realm: 35
 ct_get_entitlement_for_user: 32
 ct_get_entitlement_for_user.: 38
 ct_get_entitlement_for_user_and_appfunc: 38
 ct_get_entitlement_for_user_and_application: 38
 ct_get_entitlement_for_user_and_appurl: 38
 ct_get_exp_entitlement_for_application_by_index: 43
 ct_get_exp_entitlement_for_application_by_range: 43
 ct_get_exp_entitlement_for_group_by_index: 29
 ct_get_exp_entitlement_for_realm_by_index: 35
 ct_get_exp_entitlement_for_user_by_index: 32
 ct_get_exp_entitlements_for_group_by_range: 29
 ct_get_exp_entitlements_for_realm_by_range: 35
 ct_get_exp_entitlements_for_user_by_range: 32
 ct_get_function_table: 227
 ct_get_group_by_index: 29
 ct_get_group_by_index_in_search: 50
 ct_get_group_by_name: 29
 ct_get_group_by_name_in_search: 50
 ct_get_group_by_names_in_search: 50
 ct_get_group_for_admingroup_by_index: 21
 ct_get_group_for_admingroup_by_name: 22
 ct_get_group_for_entitlement: 39
 ct_get_group_for_realm_by_index: 35
 ct_get_group_for_realm_by_name: 35
 ct_get_group_for_user_by_index: 32
 ct_get_group_for_user_by_name: 32
 ct_get_group_owner: 29
 ct_get_groups_by_names: 29
 ct_get_groups_by_range: 29
 ct_get_groups_by_range_in_search: 50
 ct_get_groups_for_admingroup_by_names: 22
 ct_get_groups_for_admingroup_by_range: 22
 ct_get_groups_for_realm_by_names: 35
 ct_get_groups_for_realm_by_range: 35
 ct_get_groups_for_user_by_names: 32
 ct_get_groups_for_user_by_range: 32
 ct_get_num_of_admingroups: 22
 ct_get_num_of_admingroups_in_search: 50
 ct_get_num_of_adminroles_for_admingroup: 22
 ct_get_num_of_appfuncs_for_application: 43
 ct_get_num_of_applications: 43
 ct_get_num_of_applications_for_admingroup: 22
 ct_get_num_of_apps_in_search: 50
 ct_get_num_of_appurls_for_application: 43
 ct_get_num_of_appurls_for_webserver: 46
 ct_get_num_of_exp_entitlements_for_application: 43
 ct_get_num_of_exp_entitlements_for_group: 30
 ct_get_num_of_exp_entitlements_for_realm: 35
 ct_get_num_of_exp_entitlements_for_user: 32
 ct_get_num_of_groups: 30
 ct_get_num_of_groups_for_admingroup: 22
 ct_get_num_of_groups_for_realm: 35
 ct_get_num_of_groups_for_user: 32
 ct_get_num_of_groups_in_search: 50
 ct_get_num_of_password_policies: 27
 ct_get_num_of_realms: 35
 ct_get_num_of_realms_for_admingroup: 22
 ct_get_num_of_realms_for_group: 70
 ct_get_num_of_realms_in_search: 50
 ct_get_num_of_userpropdefs: 34
 ct_get_num_of_userpropdefs_for_admingroup: 22
 ct_get_num_of_userpropdefs_in_search: 50

ct_get_num_of_userprops_for_user: 32
 ct_get_num_of_users: 32
 ct_get_num_of_users_for_admingroup: 22
 ct_get_num_of_users_for_group: 30
 ct_get_num_of_users_in_group_in_search: 50
 ct_get_num_of_users_in_search: 50
 ct_get_num_of_webservers: 46
 ct_get_num_of_webservers_for_admingroup: 22
 ct_get_num_of_webservers_in_search: 50
 ct_get_num_servertree_for_webserver: 47
 ct_get_num_users_in_admin_role: 26
 ct_get_password_expiration: 19
 ct_get_password_policies_by_names: 27
 ct_get_password_policies_by_range: 27
 ct_get_password_policy_by_index: 27
 ct_get_password_policy_by_name: 27
 ct_get_realm_by_index: 35
 ct_get_realm_by_index_in_search: 50
 ct_get_realm_by_name: 35
 ct_get_realm_for_admingroup_by_index: 22
 ct_get_realm_for_admingroup_by_name: 22
 ct_get_realm_for_entitlement: 39
 ct_get_realm_for_group_by_index: 30
 ct_get_realm_for_group_by_name: 30
 ct_get_realm_owner: 36
 ct_get_realms_by_names: 36
 ct_get_realms_by_names_in_search: 50
 ct_get_realms_by_range: 36
 ct_get_realms_by_range_in_search: 50
 ct_get_realms_for_admingroup_by_names: 22
 ct_get_realms_for_admingroup_by_range: 22
 ct_get_realms_for_group_by_names: 30
 ct_get_realms_for_group_by_range: 30
 ct_get_realy_by_name_in_search: 50
 ct_get_servertree_for_webserver_by_index: 47
 ct_get_servertree_for_webserver_by_range: 47
 ct_get_servertree_owner: 47
 ct_get_token_value: 150
 ct_get_token_value_pool: 150
 ct_get_token_values: 151
 ct_get_token_values_pool: 151
 ct_get_user_and_properties: 32
 ct_get_user_and_properties_by_dn: 32
 ct_get_user_by_index: 32
 ct_get_user_by_index_in_search: 50
 ct_get_user_by_name: 32
 ct_get_user_by_name_in_search: 50
 ct_get_user_for_admingroup_by_index: 22
 ct_get_user_for_admingroup_by_name: 22
 ct_get_user_for_entitlement: 39
 ct_get_user_for_group_by_index: 30
 ct_get_user_for_group_by_name: 30
 ct_get_user_in_admin_role_by_index: 26
 ct_get_user_in_admin_role_by_name: 26
 ct_get_user_in_group_by_index_in_search: 51
 ct_get_user_in_group_by_name_in_search: 51
 ct_get_user_owner: 32
 ct_get_user_properties: 150
 ct_get_user_properties_pool: 150
 ct_get_user_property: 150
 ct_get_user_property_pool: 150
 ct_get_userprop_for_user_by_index: 32
 ct_get_userprop_for_user_by_name: 32
 ct_get_userpropdef_by_index: 34
 ct_get_userpropdef_by_index_in_search: 51
 ct_get_userpropdef_by_name: 34
 ct_get_userpropdef_by_name_in_search: 51
 ct_get_userpropdef_for_admingroup_by_index: 22
 ct_get_userpropdef_for_admingroup_by_name: 22
 ct_get_userpropdef_owner: 34
 ct_get_userpropdefs_by_names: 34
 ct_get_userpropdefs_by_names_in_search: 51
 ct_get_userpropdefs_by_range: 34
 ct_get_userpropdefs_by_range_in_search: 51
 ct_get_userpropdefs_for_admingroup_by_names: 22
 ct_get_userpropdefs_for_admingroup_by_range: 23
 ct_get_userprops_for_user_by_names: 32
 ct_get_userprops_for_user_by_range: 32
 ct_get_users_by_names: 32
 ct_get_users_by_names_in_search: 51
 ct_get_users_by_range: 32
 ct_get_users_by_range_in_search: 51
 ct_get_users_for_admingroup_by_names: 23
 ct_get_users_for_admingroup_by_range: 23
 ct_get_users_for_group_by_range: 30
 ct_get_users_for_group_by_names: 30
 ct_get_users_in_admin_role_by_names: 26
 ct_get_users_in_admin_role_by_range: 26
 ct_get_users_in_group_by_names_in_search: 51
 ct_get_users_in_group_by_range_in_search: 51
 ct_get_version: 151
 ct_get_webserver_by_index: 46
 ct_get_webserver_by_index_in_search: 51
 ct_get_webserver_by_name: 46
 ct_get_webserver_by_name_in_search: 51
 ct_get_webserver_for_admingroup_by_index: 23
 ct_get_webserver_for_admingroup_by_name: 23
 ct_get_webserver_for_appurl: 45
 ct_get_webserver_owner: 46
 ct_get_webservers_by_names: 46
 ct_get_webservers_by_names_in_search: 51
 ct_get_webservers_by_range: 46

ct_get_webservers_by_range_in_search: 51
 ct_get_webservers_for_admingroup_by_name: 23
 ct_get_webservers_for_admingroup_by_range: 23
 ct_grab_server_reference: 139
 ct_hash.h: 13, 135
 ct_init_pool_table: 139
 ct_initialize_api: 15, 18
 CT_IS_PATH_PROTECTED: 231
 ct_lock.h: 13, 135
 ct_lock_impl.h: 13, 135
 ct_login: 19
 ct_lookup_pool: 139
 ct_malloc: 229
 ct_map.h: 12, 135, 151
 ct_map_clear: 151
 ct_map_destroy: 151
 ct_map_find: 151
 ct_map_get_default_auth_result: 150
 ct_map_insert: 151
 ct_map_remove: 151
 ct_map_utils.h: 13, 135, 151
 CT_NO_AUTH_SERVERS: 232
 ct_objects.h: 13
 CT_ORIG_URI: 231
 CT_PASSWORD: 231
 CT_PasswordPolicy: 27
 ct_permissions: 52
 ct_permissions.h: 12
 CT_PLUGIN_USER: 231
 ct_pool_manager.h: 135
 CT_POST_DATA: 231
 CT_PREV_USER: 231
 ct_print
 in WAX: 229
 ct_print_pool: 139
 CT_QUERY: 231
 ct_rc_constants: 54
 ct_rc_constants.h: 12
 ct_realloc: 229
 CT_Realm
 deprecated C functions: 35
 deprecation of: 29
 ct_refresh_pool: 139
 ct_relations.h: 13
 ct_release_server_reference: 139
 CT_REMOTE_USER: 236, 253
 ct_remove_group_from_realm: 36
 ct_remove_server_from_pool: 139
 ct_remove_user_from_admin_role: 26
 ct_remove_user_from_group: 30
 ct_req_alloc: 228
 ct_req_strdup: 228
 ct_request_data: 229, 232
 ct_reset_password: 19
 ct_resource_constants.h: 135
 ct_result_constants.h: 135
 ct_return_code_keys.h: 135
 ct_revert_password: 19
 CT_ROOT
 keystore search and: 165
 Runtime Java API and: 165
 CT_RUNAPI_AUTHENTICATION_TYPE: 142
 CT_RUNAPI_CREDENTIALS_KEY: 143, 145
 CT_RUNAPI_NT_DOMAIN_KEY: 145
 CT_RUNAPI_SECURID_NEW_PIN_KEY: 143,
 146
 CT_RUNAPI_SECURID_NEXT_CODE_KEY: 144,
 147
 CT_RUNAPI_SERVER_STATE: 144
 CT_RUNAPI_TOKEN_KEY: 142
 CT_RUNAPI_USER_CERT_KEY: 141, 146
 CT_RUNAPI_USER_DN_KEY: 141
 CT_RUNAPI_USER_ID_KEY: 141, 145
 ct_runtime_api.h: 134, 149
 ct_runtime_api.jar
 DCOM and: 190
 ct_save_admin_group: 23
 ct_save_admin_role: 26
 ct_save_appfunc: 44
 ct_save_application: 43
 ct_save_appurl: 45
 ct_save_exp_entitlement: 39
 ct_save_group: 30
 ct_save_password_policy: 27
 ct_save_realm: 36
 ct_save_server_tree: 47
 ct_save_user: 32
 ct_save_user_and_properties: 33
 ct_save_userpropdef: 34
 ct_save_webserver: 46
 ct_search: 47
 ct_search.h: 12
 CT_SERVER_TIMED_OUT: 232
 CT_SESSION_ACTIVE: 232
 CT_SESSION_EXPIRED: 232
 ct_set_application_owner: 43
 ct_set_group_owner: 30
 ct_set_password: 19
 ct_set_password_expiration: 19
 ct_set_realm_owner: 36
 ct_set_servertree_owner: 47
 ct_set_token_value: 150

- ct_set_token_value_pool: 150
 - ct_set_token_values: 151
 - ct_set_token_values_pool: 151
 - ct_set_user_owner: 33
 - ct_set_userpropdef_owner: 34
 - ct_set_webserver_for_appurl: 45
 - ct_set_webserver_owner: 46
 - ct_shutdown_pool: 139
 - CT_SmartRule: 40
 - CT_SSL_CALLBACK: 140
 - CT_SSL_KEYSTORE: 140
 - CT_STATUS: 231
 - ct_strdup: 229
 - ct_structs.h: 12
 - ct_table_find: 228
 - ct_table_find (WAX method): 233
 - ct_table_put: 214, 228
 - ct_table_remove: 228
 - ct_table_replace: 228
 - ct_test_server: 151
 - ct_test_server_pool: 151
 - ct_token_keys.h: 135
 - CT_TOKENS_ENABLED: 133, 140
 - ct_transfer_ownership: 23
 - CT_UNHANDLED_REQUEST: 232
 - CT_URI: 231
 - CT_USER: 231
 - CT_User: 31
 - vs. CT_AdminUser: 24
 - ct_user_constants.h: 135
 - CT_USER_DATA: 231
 - CT_USER_DATA_LEN: 231
 - CT_USER_PROPERTIES_ENABLED: 133, 140
 - CT_UserProperty: 33
 - CT_UserPropertyDefinition: 33
 - ct_utilities: 56, 57
 - ct_utilities.h: 13, 54
 - ct_validate_token: 151
 - ct_validate_token_pool: 151
 - ct_validate_user: 19
 - ct_wax_init: 227
 - parameters to: 233
 - CT_WEB_SVR_ID: 236
 - CT_WebServer: 45
 - ct_windows.h: 135
 - CTjavaAPI: 198
 - CTLoginErrorMsg: 240
 - CTSESSION cookie name: 239
 - CTUSER: 253
 - CUSTOM
 - authentication type for WAX: 215
 - custom
 - custom parameters for WAX: 233
 - custom authentication: 215, 221
 - C Admin API vs. WAX: 148
 - C Runtime API: 148
 - custom error pages: 224
 - customauth package: 167
- ## D
- data
 - arrays in C API: 18
 - arrays in Java API: 102
 - finding with C API: 18
 - finding with Java API: 102
 - loading with C API: 18
 - loading with Java API: 102
 - dataset: 18, 102
 - date criterion: 105
 - DCOM
 - ct_dcom.jar: 190
 - jintegra_1.3.8.zip: 190
 - port number: 192
 - runvm.bat: 190
 - test.asp: 190
 - DCOM API
 - structure: 194
 - DCOM applications
 - using with RSA ClearTrust: 189
 - DCOM bridge: 189
 - DCOMFactory: 196
 - debug information
 - WAX: 229
 - defaultPrivate: 20
 - del_admin: 26
 - del_admin_user: 25
 - del_app: 25
 - del_group: 25
 - del_realm: 25
 - del_server: 25
 - del_user: 25
 - del_user_prop_def: 25
 - delete: 79, 85
 - dictionary_file: 27
 - disconnect: 70
 - Java code example: 74
 - Dispatcher
 - checking with C API: 151
 - dispatcher.list_port.force_anonymous: 136
 - distinguished name: 31
 - dll

- Administrative C API: 13
- dn: 31
- document conventions: ix
- E**
- echo
 - WAX logging: 229
- emailaddr: 31
- enddate: 31
- entitlement
 - C API: 36
 - creating in C API: 29
 - get by index in C API: 18
 - get by range in C API: 18
 - in Java API: 94
 - loading with C API: 18
 - loading with Java API: 102
- entitlements datastore
 - searching: 47
- Entitlements Server
 - vs. API Server: 1
- entity header in C API: 28
- environment variables
 - CT: 236
- error
 - custom error page with WAX: 224
 - error codes in C API: 54
 - Java Admin API error messages: 73
- example code
 - C
 - Admin API: 60
 - Admin API source files: 11
 - administrative user: 60
 - authenticate with SecurId: 153
 - ct_free_array_obj(): 56
 - ct_free_obj(): 56
 - dynamic allocation: 59
 - location, Admin examples: 12
 - location, Runtime examples: 134
 - memory management: 56
 - Runtime API: 131
 - Runtime API source files: 131
 - SecurId: 153
 - static allocation: 58
 - struct attribute memory: 57
 - DCOM
 - authenticate from ASP: 198
 - from Active Server Pages: 200
 - getting an APIServerProxy: 194
 - getting objects: 195

- user list from ASP: 201
- Java
 - Admin API: 110
 - Admin API connect: 75
 - Admin API source files: 67
 - Application and App Function: 120
 - authenticate: 172
 - authenticate with SecurId: 183
 - authenticate with SSL on: 177
 - create and edit users: 110
 - edit user property: 113
 - flushCache: 112
 - password changer: 240
 - Runtime API: 157, 172
 - Runtime API source files: 157
 - Runtime API with SSL: 177
 - SecurId: 183
 - SmartRule: 123
 - SparseData: 121
 - user search: 127
- WAX
 - authentication: 222
 - custom error page: 225
 - reporting authentication errors: 225
- excluded_chars: 27
- expired password, handling: 240
- explicit entitlement
 - C API: 36
 - creating in C API: 29
 - in Java API: 94
- export_cookie_user_buffer: 253
- export_last_touch_time: 253
- export_session_expiration_time: 253
- export_session_init_time: 253
- exportable: 133, 159
 - properties in C API: 34
- exportable property
 - enable retrieval of: 140, 159
 - enable retrieval of in C API: 133
- F**
- find data
 - C API: 18
 - Java API: 102
- firstname: 31
- float criterion: 106
- flushCache: 70
- fonts in this book: ix
- force_anonymous: 136
- forceExpiry: 20

forcePasswordExpiration: 70
 functions
 memory management: 228

G

get_apps_for_user method removed: 21
 get_default_password_policy: 27
 GET_STATUS: 232
 getAccountEndCriterion: 109
 getAccountStartCriterion: 109
 getAdminGroups: 70
 getAdministrativeGroup: 70
 getAdministrativeRole: 70
 getAdministrativeUserByUniqueIdentifier: 70
 getAdministrativeUsers: 70
 getAdministrators: 78
 getAdminRoleIdsForUser: 70
 getAllUserPropertyCriteria: 109
 getApplication: 98, 99
 getApplicationFunction: 95
 getApplicationFunctions: 96
 getApplications: 70, 78
 getApplicationURLs: 96
 getAppsForUser method removed: 70
 getBooleanValue: 105
 getByIndex: 102
 getByName: 102
 getByNames: 102
 getByRange: 102
 getCategory: 95
 getClearTrustVersion: 171
 getCreateAdministrativeRole: 82
 getCreateApplication: 82
 getCreateGroup: 82
 getCreateRealm: 82
 getCreateServer: 82
 getCreateUser: 82
 getCreateUserPropertyDefinition: 82
 getCreationDate: 89, 93
 getDateValue: 105
 getDefaultPasswordPolicy: 70
 getDeleteAdministrativeRole: 82
 getDeleteApplication: 83
 getDeleteGroup: 83
 getDeleteRealm: 83
 getDeleteServer: 83
 getDeleteUser: 83
 getDeleteUserPropertyDefinition: 83
 getDescription: 79, 85
 getDictionaryFile: 85

getDN: 89
 getDNCriterion: 109
 getEmailAddress: 89
 getEmailAddressCriterion: 109
 getEndDate: 89
 getEntity: 95
 getExclusionCharacters: 85
 getExplicitEntitlement: 70
 getExplicitEntitlements: 98
 getFirstName: 89
 getFirstNameCriterion: 109
 getFloatValue: 106
 getForceNonLetter: 85
 getGroups: 70, 78
 getHistorySize: 85
 getHost: 171
 getHostname: 100
 getIntegerValue: 106
 getLastName: 89
 getLastNameCriterion: 109
 getManufacturer: 100
 getModifyAdministrativeRole: 83
 getModifyApplication: 83
 getModifyGroup: 83
 getModifyRealm: 83
 getModifyServer: 83
 getModifyUser: 83
 getModifyUserPropertyDefinition: 83
 getName: 79, 85
 getObjectType: 105, 106
 getOperator: 105
 getOwnerCriterion: 109
 getPasswordLifetime: 85
 getPasswordMaximumLength: 85
 getPasswordMinimumLength: 85
 getPasswordPolicies: 70
 getPasswordPolicy: 78
 getPermissionChecker: 70
 getPort: 100, 171
 getPrimaryKey: 79, 85
 getPWExpirationDate: 70
 getRealms: 70, 78
 getResetPassword: 83
 getServerTrees: 100
 getSmartRuleCriteria: 95
 getSmartRules: 98
 getSocket: 70
 getStartDate: 89
 getSuperHelpDeskCriterion: 109
 getSuperUserCriterion: 109
 getTokenValue: 168

getTokenValues: 168
 getURI: 99, 101
 getUser: 70, 91
 getUserAndProperties: 70
 getUserAndPropertiesByDN: 71
 getUserByUniqueIdentifier: 71
 getUserIDCriterion: 109
 getUserLockoutCriterion: 109
 getUserProperties: 89, 168
 getUserProperty: 89, 168
 getUserPropertyDefinition: 95
 getUserPropertyDefinitions: 71, 79
 getUsers: 71, 79, 83
 getValue: 91
 getValueType: 91
 getVersion: 96
 getWebApplications: 71
 getWebApplicationURLs: 100
 getWebServer: 99
 getWebServers: 71, 79
 Global block: 233
 group
 administrative group in Java API: 78
 get by index in C API: 18
 get by range in C API: 18
 loading with C API: 18
 loading with Java API: 102
 object in C API: 29
 object in Java API: 87
 search for in Java API: 107
 See also administrative group: 20

H

handler
 augment vs. override handler: 213
 authentication: 208
 authorization: 209
 cookie: 210
 path check: 207
 phase: 205, 207
 preauthentication: 208
 session: 207
 status: 205, 231
 handler keys: 214
 hash table functions: 228
 header
 Administrative API: 134
 C Administrative API: 12
 HTTP header parameters: 253
 helpDeskAccessible: 34

history: 27
 HTTP header parameters: 253

I

IAdministrativeGroup: 78
 IAdministrativeGroupSearch: 107
 IAdministrativePermissionChecker: 103
 IAdministrativeUser: 80
 vs. IUser: 80
 IAdministrator: 82
 IAPIObject
 from SparseData: 102
 IApplication: 96
 IApplicationFunction: 97
 example code: 120
 IApplicationSearch: 107
 IGroupSearch: 107
 index
 get by index in C API: 18
 index.jsp: 241
 init_ct_runtime_api_with_ssl: 149
 input parameter request values: 230
 install
 APIs on Solaris: 9
 APIs on Windows: 7
 C Administrative API: 12, 134
 integer criterion: 106
 interfaces
 AuthTypes: 169
 CredentialConstants: 170
 ResourceConstants: 170
 ResultConstants: 170
 RuntimeAPI: 167
 TokenKeys: 169
 UserConstants: 169
 IPLANET_SERVER_DIR: 204
 IRealm
 deprecated functions: 93
 deprecation of: 87
 is_locked: 31
 isAccessible: 95
 isAdminLockedout: 89
 isDefaultPrivate: 79
 IServerTree: 101
 isExportable: 91
 isForcedPasswordExpiry: 79
 isHelpDeskAccessible: 92
 ISmartRule: 95
 ISparseData: 102, 121
 equivalent in C API: 18

isPolicyAllowBeforeDeny: 98
 isPWExpirationDateOverridden: 71
 isReadOnly: 92
 isSet: 91
 ISSO cookie
 contents: 239
 isSSLUsed: 171
 isSuperHelpDesk: 89
 isSuperUser: 89
 IUser
 vs. IAdministrativeUser: 80
 IUserPropDefSearch: 108
 IUserSearch: 109
 IWebServer: 99
 IWebServerSearch: 108

J

jars
 for DCOM: 191
 Java API
 Admin: 67
 Runtime: 157
 jce1_2-do.jar: 13
 for DCOM API: 191
 for Java API: 68, 161
 jcert.jar: 13
 for DCOM API: 191
 for Java API: 68, 161
 JCSI Keystore software
 for DCOM API: 191
 for Java API: 68, 161
 jcsi_base.jar: 13
 for DCOM API: 191
 for Java API: 68, 161
 jcsi_provider.jar: 13
 for DCOM API: 191
 for Java API: 68, 161
 J-Integra DCOM bridge: 189, 191
 jintegra_1.3.8.zip: 190
 jintegra138.zip: 191
 jnet.jar: 13
 for DCOM API: 191
 for Java API: 68, 161
 JNI wrapper: 13
 JRun: 240
 jsafe.jar: 13
 for DCOM API: 191
 for Java API: 68, 161
 jsafeJCE.jar: 13
 for DCOM API: 191

for Java API: 68, 161

JSP
 writing JSP pages: 240
 jsse.jar: 13
 for DCOM API: 191
 for Java API: 68, 161

K

key
 client keys: 141
 private key passphrase: 140
 keystore
 CT_ROOT: 165
 JCSI software for DCOM: 191
 JCSI software for Java API: 68, 161
 name: 140
 passphrase callback: 140

L

lastname: 31
 LDAP authentication
 C Runtime API: 147
 ldapattr_map_scuid: 147
 lib directory
 for DCOM API: 190
 libct_admin_api.a and .so: 13
 library
 C Admin API: 13
 C API libraries: 12, 134
 lifetime: 27
 Linar DCOM bridge: 189, 191
 link
 Agent: 216
 linking
 for Microsoft IIS: 216
 for Netscape/NT: 216
 for Netscape/UNIX: 216, 217
 list_port.force_anonymous: 136
 load data
 C API: 18
 Java API: 102
 log
 WAX information: 229
 login: 71
 login_error_password_expired: 240
 lookupUserPropertyCriterion: 109

M

malloc: 229

- Map Functions: 151
- max_length: 27
- memory
 - allocation for WAX: 228
- memory management
 - C API utilities: 54, 56
 - for WAXes: 228
- memory management functions: 228
- Microsoft Visual C++: 216
- min_length: 27
- mod_admin: 25
- mod_admin_user: 25
- mod_app: 25
- mod_group: 25
- mod_realm: 25
- mod_server: 25
- mod_user: 25
- mod_user_prop_def: 25
- multithreading: 167
- my_cookie_phase_handler: 218

N

- namespace conflicts: 217
- non_letter: 27
- NT authentication
 - C Runtime API: 145
- NT domain
 - in authentication: 145
- nt_auth.c: 221, 222
- numerical error code
 - C API: 54
- numOfUserPropertyCriteria: 109

O

- object
 - collections in C API: 18
 - collections in Java API: 102
 - get by index in C API: 18
 - get by range in C API: 18
 - loading in C API: 18
 - loading in Java API: 102
 - utilities in C API: 54
- OpenSSL: 149
- opt directory: 9
- overriding phase processing: 231

P

- package
 - customauth: 167

- sirrus.api.com: 196
- sirrus.runtime: 167
- sirrus.runtime.customauth: 167
- parameter
 - custom parameters for WAX: 233
 - HTTP header parameters: 253
 - WAX parameter scope: 233
- passphrase
 - keystore: 140
 - private key: 140
- password: 31
 - example code for changing: 240
- password policy: 20
 - object in C API: 27
 - object in Java API: 85
- PasswordChanger.java: 241
- path check handler: 207
- permission
 - checking administrator permissions in C API: 52
 - in Java API: 103
- personalization: 235
- phase handler: 205, 207
 - augment vs. override handler: 213
- phase processing
 - overriding: 231
- pix directive: 227
- PIX, [See WAX](#)
- PKCS12 keystore: 140
- pkgadd: 9
- Plug-in, [See WAX](#)
- pool
 - checking with C API: 151
- port number
 - DCOM API: 192
- preauthentication handler: 208
- print
 - WAX status information: 229
- private key
 - keystore name: 140
 - passphrase callback: 140
- processing URI requests: 205
- propArray: 31
- property
 - user: 133, 140, 159
 - user property code example: 113
- putUserPropertyCriterion: 109

R

- range
 - get by range in C API: 18

- RC_ADMINISTRATOR_NOT_FOUND: 55
 - rc_constants: 54
 - RC_INVALID_PASSWORD: 55
 - readObject: 105, 106
 - realm
 - deprecated methods in C API: 35
 - deprecated methods in Java API: 93
 - deprecation in C API: 29
 - deprecation in Java API: 87
 - redirect.c: 224, 225
 - REMOTE_USER: 236
 - removeUser: 83
 - request data: 229
 - resetAdministrativeUserPassword: 71
 - resetPassword: 71
 - resolving names: 217
 - ResourceConstants: 170
 - ResourceConstantsClass: 196
 - ResultConstants: 170
 - ResultConstantsClass: 197
 - return value
 - error number in C API: 54
 - RETURN_CODE: 170
 - revertAdministrativeUserPasswordExpirationDate: 71
 - revertPasswordExpirationDate: 71
 - role
 - admin role in Java API: 82
 - object in C API: 25
 - root directory
 - opt or other: 9
 - RSA SecurId
 - example code, C: 153
 - example code, Java: 183
 - RSA SSL
 - jars for DCOM API: 190
 - jars for Java API: 68, 161
 - RSActapi-4.7-solaris-sparc.pkg: 9
 - rsajsse.jar: 13
 - for DCOM API: 191
 - for Java API: 68, 161
 - Runtime API
 - C: 131
 - connection types: 136
 - classes: 171
 - DCOM example: 198
 - Java: 157
 - connection types: 162
 - Runtime Functions: 149
 - Authorization and Authentication: 149
 - SSO Tokens: 150
 - User Properties: 150, 151
 - RuntimeAPI interface: 167
 - methods of: 168
 - RuntimeExample.java: 172
 - RuntimeSSLExample.java: 177
 - runvm.bat: 190, 191
- ## S
- samples directory
 - C Admin: 12, 134
 - save: 79, 85
 - SC_AUTH_TYPE_BASIC: 145
 - SC_AUTH_TYPE_CERT: 146
 - SC_AUTH_TYPE_CUSTOM: 148, 215
 - SC_AUTH_TYPE_LDAP: 147
 - SC_AUTH_TYPE_NT: 145
 - SC_AUTH_TYPE_SECURID: 146
 - SC_AUTH_TYPE_USER_CHECK: 143
 - SC_TOKENS_ENABLED: 159
 - SC_USER_PROPERTIES_ENABLED: 160
 - scope
 - of WAX parameters: 233
 - search
 - C API search tools: 49
 - entitlements datastore: 47
 - searchAdministrativeGroupObjects: 71
 - searchAdministrativeUserObjects: 71
 - searchGroupObjects: 71
 - searchRealmObjects: 71
 - searchUserObjects: 71
 - searchUserPropDefObjects: 71
 - searchWebServerObjects: 71
 - SecurantDCOMFactory: 196
 - SecurId
 - example code , C: 153
 - example code, Java: 183
 - SecurID authentication
 - C Runtime API: 146
 - server
 - API Server: 1
 - Entitlements Server: 1
 - server tree
 - object in C API: 46
 - object in Java API: 101
 - ServerDescriptor: 171
 - ServerDescriptor methods: 171
 - service, getting: xi
 - session data
 - insert into cookie: 218
 - session handler: 207

session token: 133, 159
 set_default_password_policy: 27
 set_password: 26
 SET_STATUS: 232
 setAccessible: 95
 setAccountEndCriterion: 109
 setAccountStartCriterion: 109
 setAdminLockedout: 89
 setBooleanValue: 105
 setCategory: 95
 setCreateAdministrativeRole: 83
 setCreateApplication: 83
 setCreateGroup: 83
 setCreateRealm: 84
 setCreateServer: 84
 setCreateUser: 84
 setCreateUserPropertyDefinition: 84
 setDateValue: 105
 setDefaultPasswordPolicy: 71
 setDeleteAdministrativeRole: 84
 setDeleteApplication: 84
 setDeleteGroup: 84
 setDeleteRealm: 84
 setDeleteServer: 84
 setDeleteUser: 84
 setDeleteUserPropertyDefinition: 84
 setDescription: 79, 85
 setDefaultPrivate: 79
 setDictionaryFile: 85
 setDN: 89
 setDNCriterion: 109
 setEmailAddress: 89
 setEmailAddressCriterion: 109
 setEndDate: 89
 setExclusionCharacters: 85
 setExportable: 91
 setFirstName: 89
 setFirstNameCriterion: 109
 setFloatValue: 106
 setForcedPasswordExpiry: 79
 setForceNonLetter: 85
 setHelpDeskAccessible: 92
 setHistorySize: 85
 setHostname: 100
 setIntegerValue: 106
 setLastName: 89
 setLastNameCriterion: 109
 setLengthParams: 85
 setManufacturer: 100
 setModifyAdministrativeRole: 84
 setModifyApplication: 84
 setModifyGroup: 84
 setModifyRealm: 84
 setModifyServer: 84
 setModifyUser: 84
 setModifyUserPropertyDefinition: 84
 setName: 79, 85
 setOperator: 105
 setOwnerCriterion: 109
 setPassword: 71, 89
 setPasswordLifetime: 85
 setPasswordPolicy: 79
 setPolicyAllowBeforeDeny: 98
 setPort: 100
 setPWExpirationDate: 71
 setReadOnly: 92
 setResetPassword: 85
 setSmartRuleCriteria: 95
 setStartDate: 89
 setSuperHelpDesk: 89
 setSuperHelpDeskCriterion: 109
 setSuperUser: 89
 setSuperUserCriterion: 109
 setTimeout: 71
 setTokenValue: 168
 setTokenValues: 168
 Setup.exe: 7
 setURI: 99, 101
 setUserIDCriterion: 109
 setUserLockoutCriterion: 109
 setUserProperty: 90
 setValue: 91
 setValueType: 91
 setVersion: 96
 setWebServer: 99
 SF_NOTIFY_URL_MAP: 230
 single sign-on token: 133, 159
 sirus.api.client.UserNotAuthorizedException: 73
 sirus.api.com: 196
 sirus.runtime.customauth: 148
 SmartRule
 example code: 123
 loading with C API: 18
 loading with Java API: 102
 object in C API: 40
 object in Java API: 95
 user property and: 33
 SmartRuleExample.java: 123
 Solaris
 installation of APIs: 9
 sparse data: 102
 SSL

- Admin C API connection types: 17
 - Admin Java API connection types: 72
 - jars for DCOM API: 190
 - jars for Java API: 68, 161
 - key passphrase: 140
 - keystore passphrase: 140
 - libraries: 13
 - running C Runtime API with: 136
 - Runtime C API connection types: 136
 - Runtime Java API connection types: 162
 - software for DCOM API: 190
 - software for Java API: 68, 161
 - SSL callback: 140
 - SSL keystore: 140
 - sslj.jar: 13
 - for DCOM API: 191
 - for Java API: 68, 161
 - SSO
 - cookie contents: 239
 - cookie name requirements: 240
 - SSO Token: 133, 159
 - keys in: 169
 - startdate: 31
 - status codes: 232
 - status handler: 205, 231
 - string copy
 - for WAX: 228
 - string dup: 229
 - structs: 57
 - success.jsp: 241
 - Sun
 - SSL jars for DCOM API: 191
 - SSL jars for Java API: 68, 161
 - superHelpDesk: 31
 - superuser: 31
 - support
 - contact information: xi
- T**
- test
 - test Authorization Server with C API: 151
 - test.asp: 190
 - testServer: 168
 - token: 133, 159
 - authenticated SSL: 159
 - contents: 239
 - enabling: 140, 159
 - enabling in C Runtime API: 133
 - SSO: 159
 - SSO token in C Runtime API: 133
 - TokenKeys: 169
 - trace
 - WAX logging: 229
 - transfer_admingroup: 27
 - transferOwnership: 79
 - typefaces in this book: ix
- U**
- unix
 - installation of APIs: 9
 - URI: 99
 - authentication type for: 207
 - object in C API: 44, 99
 - URI requests
 - processing: 205
 - URL
 - object in Java API: 99
 - use_ssl
 - in C API: 17
 - user
 - administrative user in C API: 24
 - administrative user in Java API: 80
 - cookie contents: 239
 - example code for ASP and DCOM: 201
 - example Java code to create users: 110
 - example user search in Java: 127
 - get by index in C API: 18
 - get by range in C API: 18
 - group of users in Java API: 87
 - groups in C API: 29
 - loading with C API: 18
 - loading with Java API: 102
 - mapping to external user name: 31
 - object in C API: 31
 - object in Java API: 89
 - search for in Java API: 109
 - searching for with Java API: 127
 - token: 133
 - user property in C API: 133
 - user property
 - denying access to: 159
 - denying access to in C API: 133
 - enabling retrieval: 133, 140, 159
 - example Java code to edit user property: 113
 - exportable: 159
 - exportable in C API: 133
 - hiding from API users: 34
 - in C API: 33
 - object in Java API: 91
 - user property definition

- in C API: 33
- object in Java API: 91
- search for in Java API: 108
- user_header_list: 253
- UserConstants: 169
- UserConstants interface: 169
- UserConstantsClass: 197
- UserNotAuthorizedException: 73
- UserPropertyTypesClass: 197
- utility
 - C Admin utilities: 54
 - Java Admin utilities: 102

V

- validateUser: 71
- vbu: 20
- virtual business unit
 - in Java API: 20
- virtual host
 - WAX and: 233
- Visual C++: 216

W

- WAX: 203
 - allocating memory: 228
 - API: 203
 - augment vs. override handler: 213
 - chaining: 205
 - compiling: 216
 - custom authentication: 215
 - custom parameters: 233
 - error page example: 225

- example: 222
- handler flow diagram: 211
- initializing Runtime API: 149
- loading and initializing: 227
- process flow diagram: 206
- registering a WAX method: 214
- registering a WAX program: 212, 213
- string duplication: 228
- virtual host and: 233
- wax directive: 227
- wax.c: 218
- WAX API: 203
 - location: 212
- Web Agent
 - edit cookie: 218
- Web Agent Extension API, [See WAX](#)
- Web Server
 - object in C API: 45
 - object in Java API: 99
 - search for in Java API: 108
- Web Server Agent
 - libraries: 213
- webagent.conf
 - auth_resource_list: 207
 - custom WAX parameters: 233
 - HTTP header parameters: 253
 - parameter scope: 233
 - WAX scoping: 233
- websrvr_id: 44
- WinZip: 241
- wrapper, JNI: 13
- writeObject: 105, 106